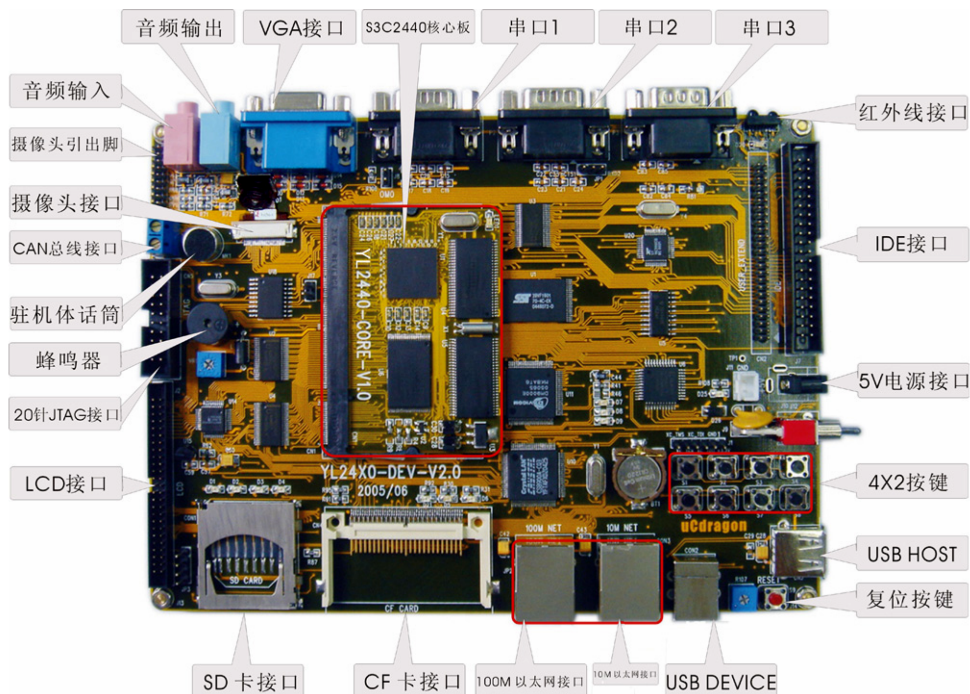
YL2440 使用手册 V2.2

第一章 YL2440 开发板套件介绍

1.1. YL2440 开发板简介

YL2440 开发板外观:



YL2440 开发板硬件资源:

中央处理器

—— CPU: 三星 S3C2440A，主频 400MHz;

外部存储器

—— SDRAM：64M 字节;

—— NOR Flash：2M 字节（SST39VF160 或 SST39VF1601）;

—— NAND Flash：64M 字节（K9F1208，用户可自己更换为 16M、32M 或 128M 的

NandFlash）

串口

—— 两个五线异步串行口，波特率高达 115200bps；

—— 一个九线异步串行口，采用 ST16C550 扩展出来的，波特率高达 1.5Mbps；

网络接口

—— 一个 10M 网口，采用 CS8900Q3，带联接和传输指示灯；

—— 一个 100M 网口，采用 DM9000，带联接和传输指示灯；

USB 接口

—— 一个 USB1.1 HOST 接口；

—— 一个 USB1.1 Device 接口；

红外通讯口

—— 一个 IRDA 红外线数据通讯口；

CAN 总线接口

—— 一个 CAN 总线接口，全面支持 CAN2.0A 和 CAN2.0B 协议；

音频接口

—— 采用 IIS 接口芯片 UDA1341，一路立体声音频输出接口可接耳机或音箱；

—— 支持录音，板子自带驻机体话筒可直接录音，另有一路话筒输入接口可接麦克风；

存储接口

—— 一个 SD 卡接口，可接 256M SD 卡；

—— 一个 CF 卡接口（3.3V，接口信号均加了 74LVTH162245 驱动），工作在 TrueIDE 模式；

—— 一个 IDE 接口(接口信号均加了 74LVTH162245 驱动)，可直接挂接硬盘；

LCD 和触摸屏接口

—— 板上集成了 4 线电阻式触摸屏接口的相关电路；

—— 一个 50 芯 LCD 接口引出了 LCD 控制器的全部信号，并且这些信号引脚都加了 74LVTH162245 驱动，所以 LCD 输出更加稳定可靠；

—— 支持黑白、4 级灰度、16 级灰度、256 色、4096 色 STN 液晶屏，尺寸从 3.5 寸到 12.1 寸，屏幕分辨率可达到 1024×768 象素；

—— 支持黑白、4 级灰度、16 级灰度、256 色、64K 色、真彩色 TFT 液晶屏，尺寸从 3.5 寸到 12.1 寸，屏幕分辨率可达到 1024×768 象素；

—— 标准配置为夏普 256K 色 240x320/3.5 英寸 TFT 液晶屏，带触摸屏；

—— 板上引出一个 5V 电源输出接口，可为大尺寸 TFT 液晶屏的 5V CCFL 背光模块供电；

摄像头接口

—— 板上自带一个 130 万象素的摄像头，可直接摄像并在液晶屏上显示，并有一个 2 毫米间距双排插座用作摄像头扩展，用户可使用这个扩展口连接其他型号摄像头；

VGA 接口

—— 一个标准 VGA 接口，可直接连接各种 VGA 接口的 CRT 显示器或液晶显示器，带对比度**微调电位器；**

时钟源

—— 内部实时时钟（带有后备锂电池）；

复位电路

—— 一个复位按键，并采用专用复位芯片进行复位，稳定可靠；

调试及下载接口

—— 一个 20 芯 Multi－ICE 标准 JTAG 接口，支持 SDT2.51,ADS1.2 等调试 ；

电源接口

—— 5V 电源供电，带电源开关和指示灯；

IIC 接口

其他

—— 八个小按键，四个高亮 LED；

—— 一个蜂鸣器（带使能控制的短路块）；

—— 一个可调电阻接到 ADC 引脚上用来验证模数转换；

—— 一个 50 芯 2 毫米间距双排标准连接器用作扩展口，引出了地址线、数据线、读写、片选、中断、IO 口、ADC、5V 和 3.3V 电源、地等用户扩展可能用到的信号；

操作系统

支持 Linux2.4.18、WinCE4.2

用户光盘上提供的开发工具和源代码：

1) ADS1.20 安装程序(评估版)；
2) 使用 SUPERJTAG 并支持 ADS1.20 的 JTAG 调试软件 ARMJTAGDEBUGFINAL；
3) 烧写 FLASH 的工具软件 SJF2440（包含 NT/2000/XP 解决方案）；
4) 串口工具软件 sscom32.exe、dnw.exe、tftp.exe；
5) 64K 色（RGB565）图片字模软件；
6) USB Device 接口驱动程序；

7) BIOS 源代码（ADS1.20 的项目文件）；

8）测试程序(ADS1.20 的项目文件，包含全部源代码)，包括如下测试：

RTC 实时时钟测试、按键测试（中断测试）、蜂鸣器测试、SD 卡读写测试、蜂鸣器测试（PWM 测试）、ADC 模数转换器测试、IIS 音频播放 wav 音乐测试、IIS 音频录音测试、IrDA 红外测试、触摸屏测试、3.5 寸夏普 TFT 液晶屏测试、CAN 总线测试、VGA 测试、摄像头测试等等；

9) Linux for S3c2440 内核源码包以及编译工具，含 CS8900 和 DM9000 的 EHTNENET 端口驱动，UART 驱动 USB HOST & DEVICE 驱动；

10) 核心板和底板电路原理图（pdf 格式）；

11) 开发板使用手册（pdf 格式）；

12) 开发板上所用到的全部芯片手册、资料；

13）ADS 使用、DNW 串口使用和超级终端配置的一些多媒体演示；

YL2440 开发板套件

1) 一块已测试好的 YL2440 开发板（包括 YL2440 核心板与底板）

2) YL2440 用户光盘

3）3.5" TFT 彩色 LCD 板一块 ，带触摸屏

4）一个 SUPER JTAG 调试头（带 20 芯排线）

5）一条并口线（一边是公头一边是母头，一对一）

6）一条串口线(两边都是母头，交叉串口线)

7）一条网线(交叉网线)

8) USB 线一条

9) 触摸笔一支

10) 一个＋5V 直流电源

11）一个包装盒

1.2. 硬件资源分配

1.2.1. 地址空间分配以及片选信号定义

S3C2440 支持两种启动模式：一种是从 NAND FLASH 启动；

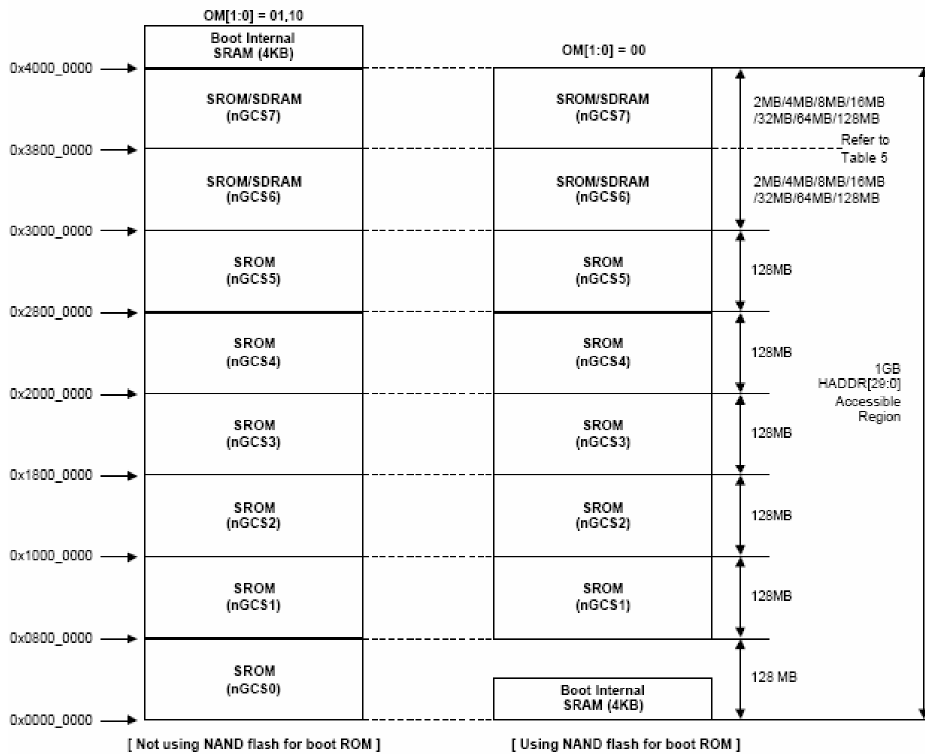
一种是从外部 nGCS0 片选的 Nor Flash 启动

启动模式的选择，主要是通过 J8（OM0）跳线座来决定的。

J8 接上跳线帽，从 NAND FLASH 启动。

J8 不接上跳线帽，从 NOR FLASH 启动。

在这两种启动模式下，各片选的存储空间分配是不同的，这两种启动模式的存储分配图如下：



a)图是 nGCS0 片选的 Nor Flash 启动模式下的存储分配图；

b)图是 NAND FLASH 启动模式下的存储分配图；

说明：SFR Area 为特殊寄存器地址空间

◆ 下面是器件地址空间分配和其片选定义

在进行器件地址说明之前，有一个点需要注意，nGCS0 片选的空间在不同的启动模式下，映射的器件是不一样的。由上图可以知道：

- 在 NAND FLASH 启动模式下，内部的 4K Bytes BootSRam 被映射到 nGCS0 片选的空间。
- 在 Nor Flash 启动模式（非 NAND FLASH 启动模式）下，与 nGCS0 相连的外部存储器 Nor Flash 就被映射到 nGCS0 片选的空间

SDRAM 地址空间：0x30000000~0x34000000

1.2.2. 跳线说明

◆ 跳线分配表

跳线名称	说明
J8（底板）	决定 S3C2440 的启动模式 **插上短路块从 Nand Flash 中启动，默认** **不插上短路块从 Nor Flash（SST39VF1601）中启动**
JP1	短路块决定 RXD2 信号是 UART2 功能还是 IrDA 功能。 短路帽接在 1，2 脚时,决定工作在 UART 功能，(默认) 短路帽接在 2，3 脚时,决定工作在 IrDA 功能；
J17	选择 CAN 总线的匹配电阻，根据用户需要

1.2.3. 接口说明

接口名称	说明
JP4	摄像头引出脚
J6	临近 JP4 的两芯大接头为 CAN 总线接口
JTAG（CN5）	20 针
T1（RJ45）	以太网接口（RJ45，带隔离器的）
UART1（J8），UART2（J7）	串行口 1，2

U3	红外线 IrDA
SD_CARD(J18)	SD 卡接口
CF_CARD	CF 卡接口
100M NET	100M 以太网接口
10M NET	10M 以太网接口
CON2	USB DEVICE 接口
R107	AIN0 的电位调节器
CN3	USB HOST 0 接口
USB_DEVICE(J17)	USB DEVICE 接口
J10	电源接口
J7	IDE 硬盘接口
CN3	用户扩展接口
U8	红外接口
P3	串口 3(16550 外扩串口)
P2	串口 2(CPU 串口 1 和串口 2)
P1	串口 1(CPU 串口 0)
J3	VGA 接口
J5	音频输出接口
J4	音频输入接口
J2	LCD 接口
JP3	触摸屏接口

1.3. 操作系统支持的驱动

YL2440 开发板支持 WINCE 和 Linux 嵌入操作系统。

- 嵌入 Linux 操作系统说明：
  - 采用 linux2.4 以上的内核
  - 支持多种文件系统,象 CRAMFS，FAT 以及用于 NAND FLASH 的 YAFFS 文件系统等等

- 支持 LCD 和触摸屏
- 支持 USB HOST
- 支持 QT
- 支持 MP3 播放和视频播放
- 支持多种网络应用，象 FTP，HTTP，Telnet 之类的网络应用
- 两个以太网口，一个 10M，一个 100M

◆ Wice 操作系统说明：

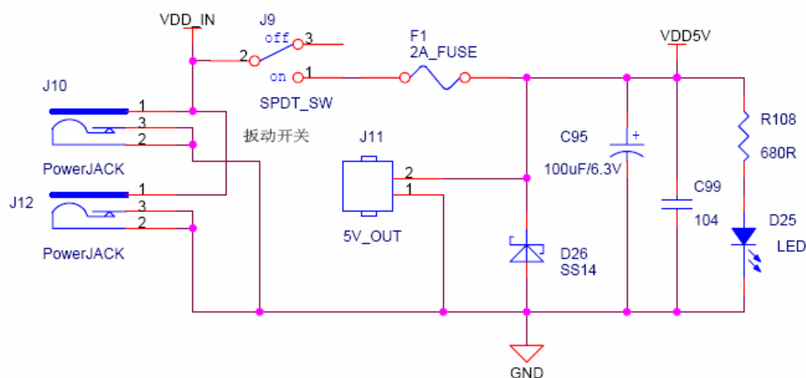
- WINCE 4.2
- 支持 SD 卡系统等等
- 支持 LCD 和触摸屏
- 支持 USB HOST
- 支持音频
- 以太网驱动（CS8900）

第二章 YL2440 开发板主要硬件描述

2.1 电源电路

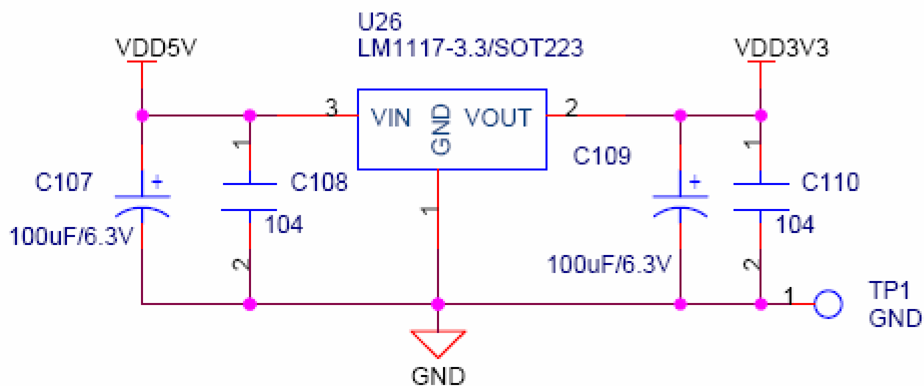
2.1.1 总电源

这一部分电源主要是从外部稳压电源输入 5V 电源，对整个板供电。



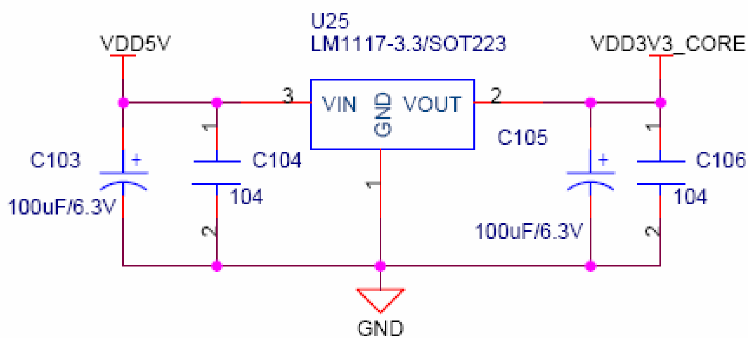
2.1.2 底板通用 3.3V 电源

主要是对底板上 3.3V 器件进行供电。



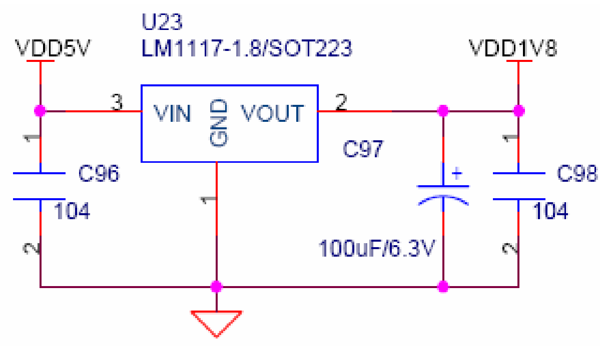
2.1.3 核心板单独供电电源

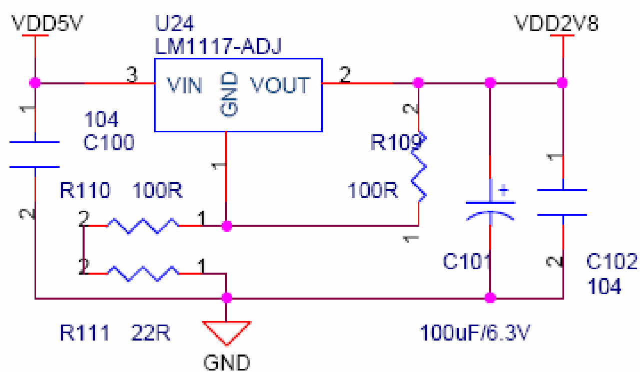
本开发板为确保核心板稳定运行，对核心板进行了单独供电。



2.1.4 摄像头供电电源

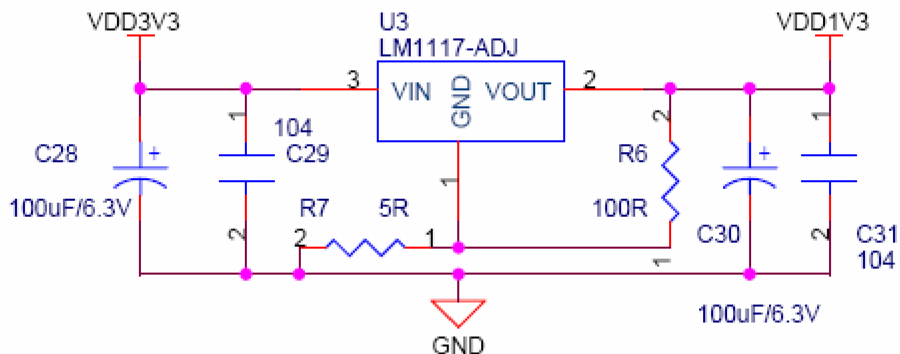
摄像头工作需要三组电源 3.3V 2.8V 1.8V





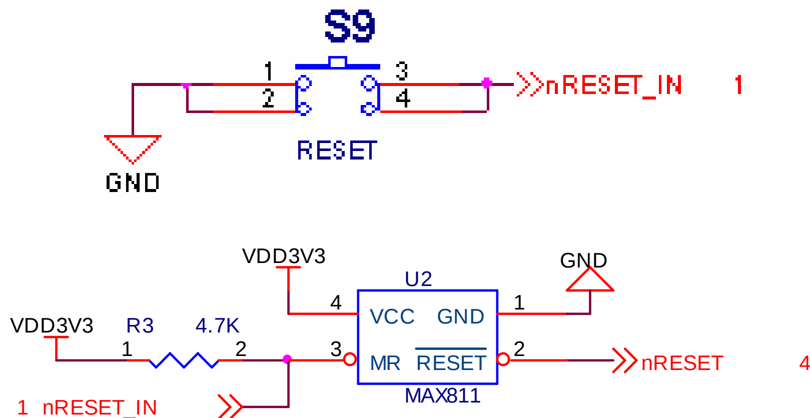
2.1.5 核心板 CPU 供电电源

S3C2440A 需两路电源，3.3V 和 1.3V。(请注意,此图网络线标示与母板标示不同，请区别对待)



2.2 复位电路

为了提高可靠性，复位电路没有采用简单的阻容复位电路，而是采用了专用复位芯片，具体电路如下：

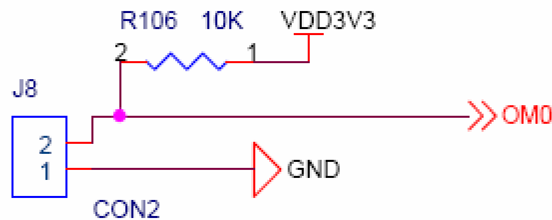


2.3 启动方式选择电路

通过跳线设置，可以选择 Nand Flash 启动和 Nor Flash 启动两种方式。

J8 接上跳线帽，从 NAND FLASH 启动。

J8 不接上跳线帽，从 NOR FLASH 启动。

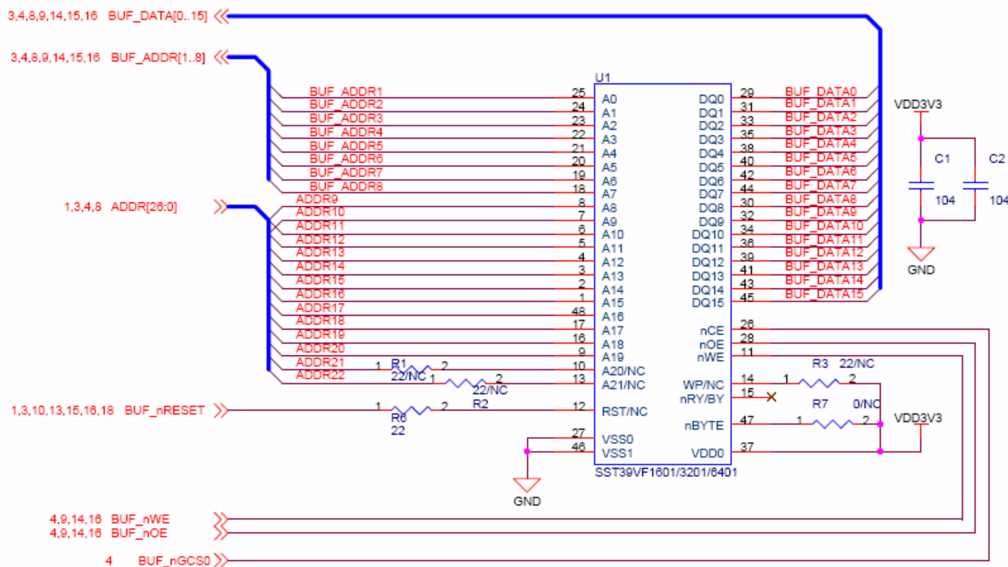


2.4 核心板上的 SDAM 电路

考虑到用户的实际需要，SDRAM 电路采用了兼容设计的方案，支持 16M 字节或 32 字节的单片 SDRAM，由如图中的两个电阻进行选择。如果是 16M 的 SDRAM（如 K4S281632）则将 R8 的 2,3 脚用 0Ω 电阻连接，R9 的 2,3 脚用 0Ω 电阻连接。如果焊的是 32M 的 SDRAM（如 K4S561632），则将 R8 的 1 , 2 脚用 0Ω 电阻连接，R9 的 1 , 2 脚用 0Ω 电阻连接。相关电路如下：

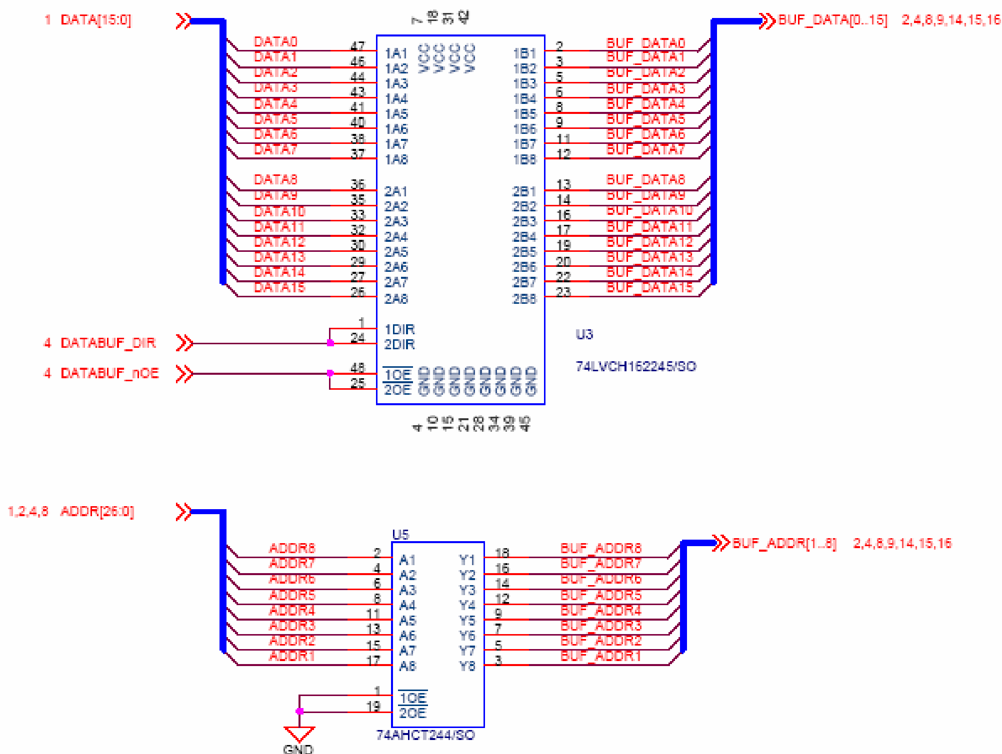
2.5 Nor Flash 电路

本开发板将 Nor Flash 放在了底板上主要是考虑核心板体积问题和用户实际应用问题，由于低地址线和数据总线要驱动多个器件，在设计时加了一级 Buffer 进行驱动。在使用过程中，请注意插拔核心板，确保核心板与座的良好接触。考虑到 NORFLASH 的货源和价格因素，NORFLASH 电路也采用了兼容设计方案，根据其周围焊接不同的电阻，可以支持多种不同型号与容量的 NORFLASH，具体电路如下：



2.6 数据地址缓冲电路

数据地址缓冲电路如下：



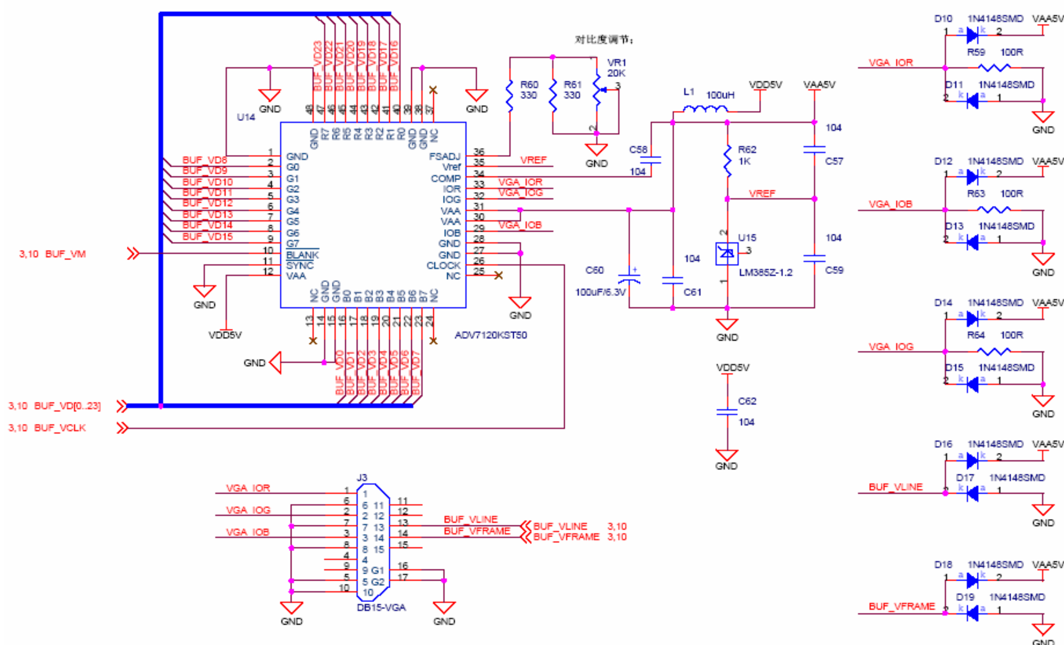
2.7 CPLD 电路

YL2440 开发板上的 CPLD 主要是实现一些控控制信号的缓冲和外部器件的一个片选分配，具体功能请参看光盘中的 CPLD 相关文档。

U6
XC9536XL
1 nGCS1 27 I/O
1 nGCS2 28 I/O
1 nGCS3 29 I/O
1 nGCS5 30 I/O
1 nGCS7 31 I/O
3 DATABUF_DIR 32 I/O
3 DATABUF_nOE 37 I/O
8,9,14 RESET 38 I/O
1 nOE 39 I/O
1 nWE 40 I/O
2,9,14,16 BUF_nOE 41 I/O
2,9,14,16 BUF_nWE 42 I/O
1 CLKOUT0 43 I/O/GCK1
1 CLKOUT1 44 I/O/GCK2
1 nGCS0 1 I/O/GCK3
3,18 nRESET 33 I/O/GSR
ADDR23 36 I/O/GTS1
ADDR24 34 I/O/GTS2
1,2,3,8 ADDR[26:0]
XC_TDO 24 TDO
35 15 VCC VCC
GND GND GND VCCIO 26 VDD3V3
I/O 2 LAN_nIOR 8
I/O 3 LAN_nIOW 8
I/O 5 CS8900_nCS 8
I/O 6 DM9000_nCS 9
I/O 7 IDE_DIOW 15
I/O 8 IDE_DIOR 15
I/O 12 CF_nCS0 16
I/O 13 CF_nCS1 16
I/O 14 CPLD_USER1 ST16C550_CS 14
I/O 16 CPLD_USER2
I/O 18 CPLD_USER3
I/O 19 CPLD_USER4
I/O 20 CPLD_USER5
I/O 21
I/O 22 BUF_nGCS0 2
I/O 23 nGCS4 1
TDI 9 XC_TDI
TCK 11 XC_TCK
TMS 10 XC_TMS



2.8 VGA 输出电路



其它电路请参看光盘中的原理图文件，这里不再进行说明。

第三章 **YL2440** 开发板使用

3.1. 开发板设置及连接

在出厂前，已将 Linux 烧写到板子上了，这一章节的工作环境是在 WINDOWS。

3.1.1. 启动模式选择

启动模式的选择，前面面已介绍本开发板是通过 J8（OM0）跳线座来决定的。

J8 接上跳线帽，从 NAND FLASH 启动。

J8 不接上跳线帽，从 NOR FLASH 启动。

出厂时开发板的起动代码置于 NAND FLASH 中，所以 J8 应处于短接状态。

3.1.2. 其它跳线设置

JP1 是选择 CUP 中的串口 2(不是指板上的串口 2)中 RXD2 是作为 nCTS1 还是 RXD2,没有用到红外端口(IrDA)时，JP1 的 1,2 脚短接。

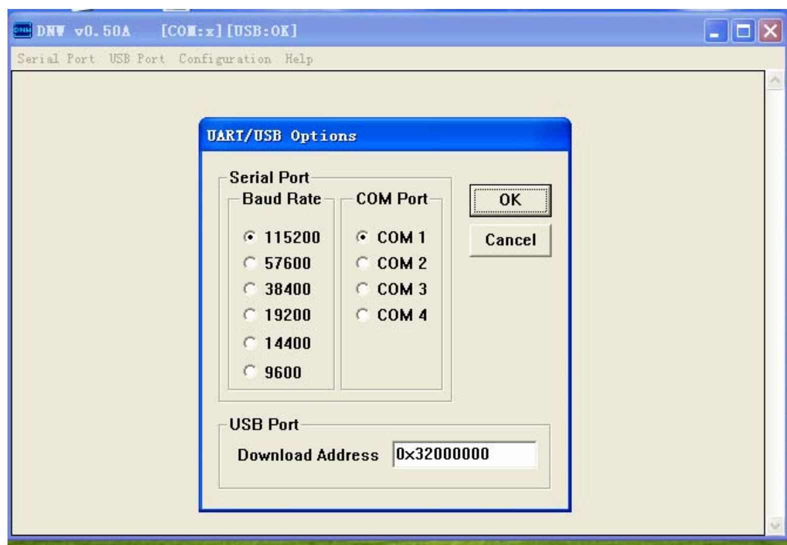
J17 是选择 CAN 总线的匹配电阻(120Ω)，根据用户需要自行选择，近距离通讯时可以不连接。

3.1.3. 外部硬件连接

1．用交叉串口连接线（开发套件中提供）将开发板上的串口 P2 与 PC 机串口 1 相连
2．用交叉网线（一头为 A 型接法一头为 B 型接法）将 CON3（10M NET）与 PC 机相连
3．用 USB 连接线将开发板与 PC 机相连
4．5V 电源连接到开发板上
5．有液晶屏的话连接上液晶屏和触摸屏（J2 和 JP3）
6．音箱输入连接到 J5 音频输出接头（绿色座）

3.1.4. 调试终端配置

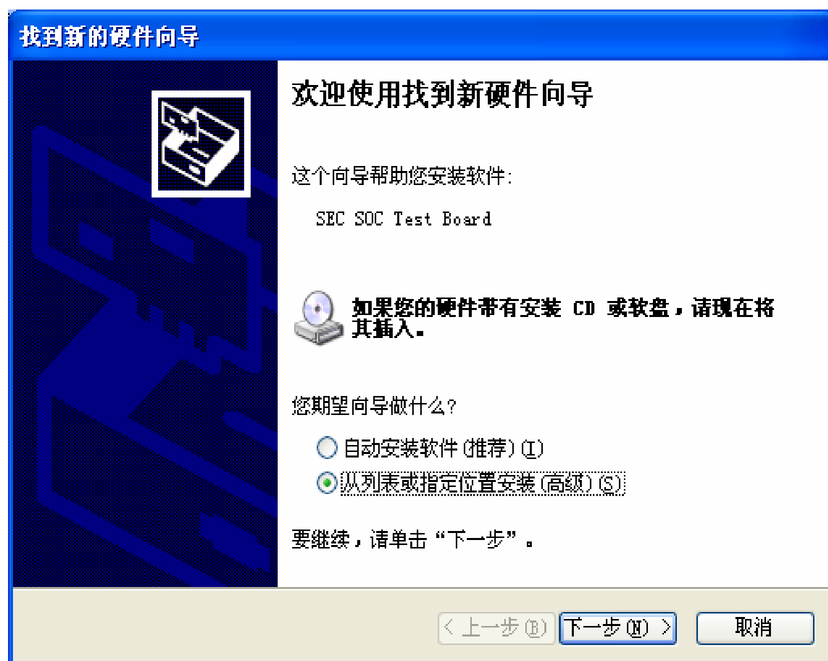
在 PC 机上 DNW(0.50)，在 configuration 中设置为：COM1（根据情况自己选定串口），波特率为 115200，8 位，无奇偶位，停止位 1，无硬件流。具体配置可参看本开发板提供的多媒体演示。也可用超级终端进行操作，具体使用请看本开发板提供的多媒体演示。



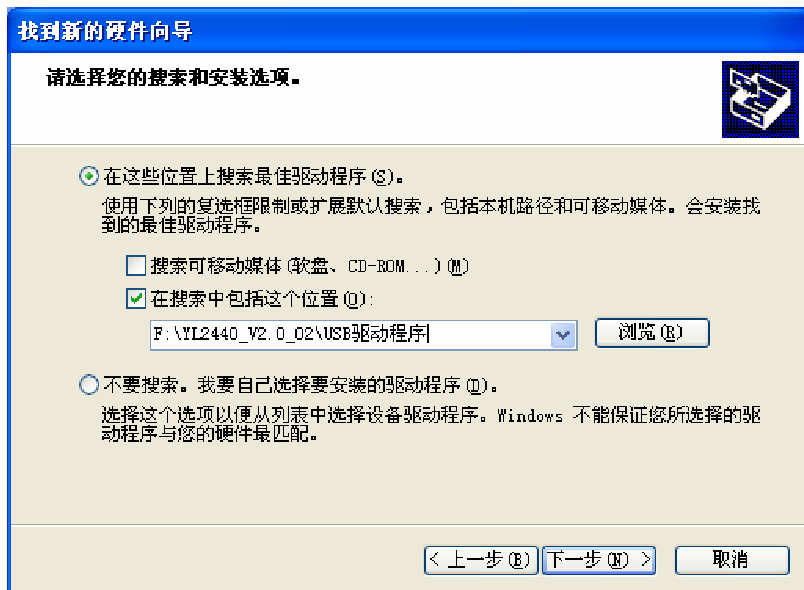
3.2. 开发板开机使用

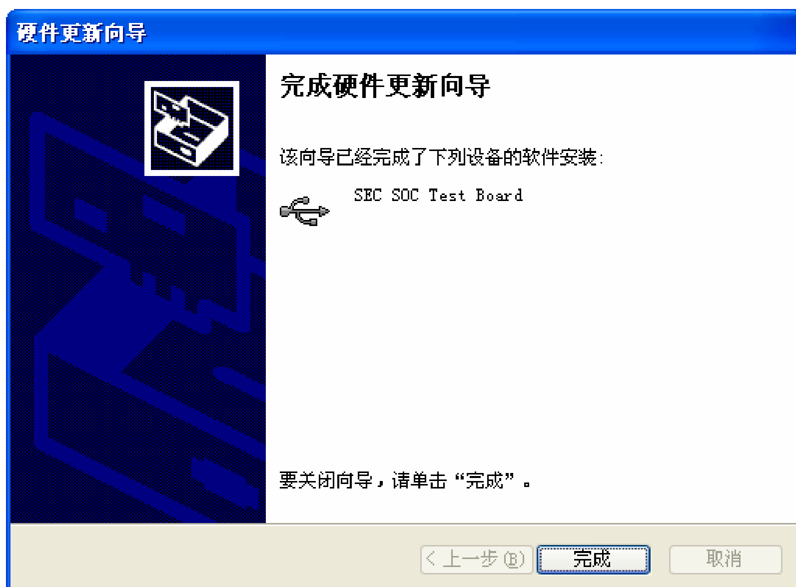
打开电源，系统会提示找到新硬件，按以下步骤安装好 USB 驱动：

1． 出现以下提示，选择“从列表或指定位置安装...”

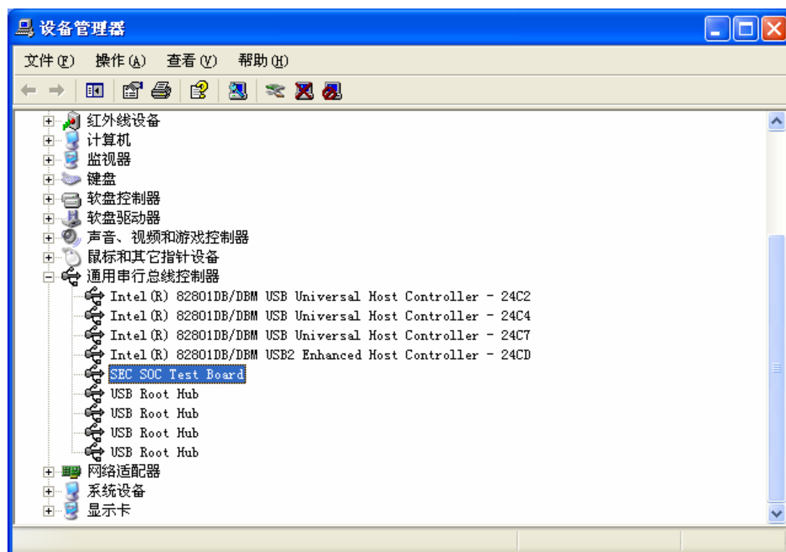


点击下一步，选择开发板提供的光盘所提供的驱动路径，点击下一步。





安装好 USB 驱动后可以看到电脑的硬件中多出了如下硬件：



现在在 DNW 中已经可以看到很多信息，在出厂前，设置 Linux 自启动，开发板上电后，BootLoader 等待一段时间，如果没有输入，将会自动启动这个 linux 系统，这时将在串口

和 LCD 屏（选配模块）有信息显示，在串口，将进入 Linux 的命令模式下，在 LCD 屏上，将有 QT 界面的显示，要进入 QT，先要进行触摸屏的校准后，一路 NEXT 下去，才能进入 QT 的界面。下图是在超级终端中能看到的启动 Linux 后的信息：

```
115200 - 超级终端
文件(F) 编辑(E) 查看(V) 呼叫(C) 传送(T) 帮助(H)

Please press Enter to activate this console. /bin/cp: will not create hard link
'/tmp/Applications/Network/modules' to directory '/tmp/Applications/AlarmServer'

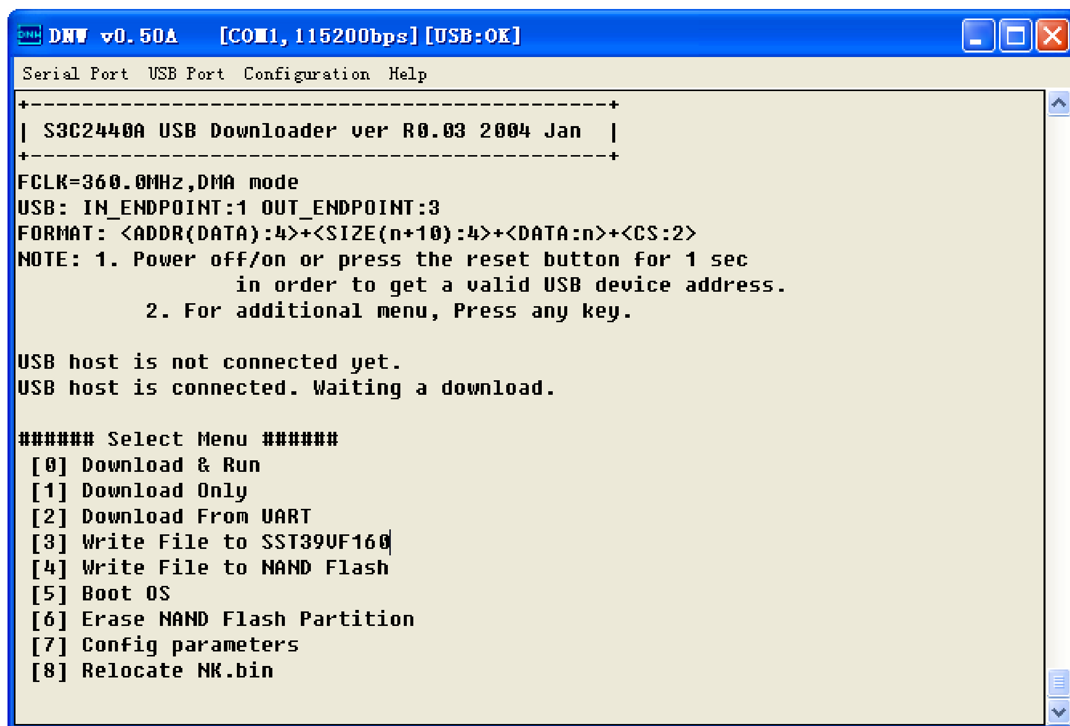
/bin/cp: will not create hard link '/tmp/Documents' to directory '/tmp/Applicati
ons/qimpen'
Could not open mime.types file, "etc/mime.types", for reading
Could not read calibration: /etc/pointercal
Create pluginlibman in libqpe
Use QPEApplication's PluginLibraryManager
Setting up QCop to QPE/System
channel QPE/Card added
channel QPE/Server added
channel QPE/IME added
Create pluginlibman in libqpe
Use QPEApplication's PluginLibraryManager
QuickLauncher running
channel QPE/Application/quicklauncher added
channel QPE/QuickLauncher-63 added
Registered QPE/QuickLauncher-63 敲回车

ln: '/dev/touchscreen/0': File exists
[root@(none) /]# ls
Qtopia  bin  dev  etc  home  lib  linuxrc  proc  ramdisk  sbin  tmp  usr  var
[root@(none) /]#

已连接 0:01:52 ANSIW  115200 8-N-1  SCROLL  CAPS  NUM  捕  打印
```

3.3. YL2440 的初步使用

按复位键后，在 DNW 中敲入任意键进入 BIOS 启动目录界面如下图:



3.3.1. YL2440 的 BIOS 使用

下面是 BIOS 功能说明。

YL2440 开发板所带的 BIOS 主要功能有文件下载（USB 和串口）、FLASH 烧写（NOR 和 NAND）、启动存储在 FLASH 中的程序、设置启动参数（针对 LINUX ）。

BIOS 菜单的 0 和 1 号功能分别是 USB 和串口下载，都是配合 DNW 这个程序使用的，在 DNW 上的 Configuration 菜单的 Options 选项里可以选择 PC 和开发板同讯的串口和波特率，还可以指定 USB 下载时开发板要将下载的文件保存到 SDRAM 中的起始地址，对于 YL2440 开发板，SDRAM 的物理起始地址是 0x30000000，结束地址是 0x34000000，大小是

64Mbytes，所以指定的 USB 下载地址要在此范围内。另外 BIOS 自身占用了 0x30200000 以下的空间，因此在用 BIOS 的 USB 下载功能时应指定地址大于等于 0x30200000。用串口下载时，BIOS 内部固定了下载地址为 0x30400000。BIOS 在下载结束后，会询问是否执行下载的程序，利用此功能可引导 LINUX 和 WINCE 及用户自编的应用程序。注意用 USB 下载要先在 PC 端装好驱动程序，保证 USB 连接好，有时 PC 端出现发现无法识别的 USB 设备时，可在 BIOS 中输入 ESC 取消下载，等几秒钟再输入 0 启动 USB 下载。

BIOS 的 2 号功能可执行 NAND FLASH 的烧写，目前支持三星的 32M 和 64M 两种型号。BIOS 对 NAND FLASH 作了简单分区，主要是为适应 LINUX 操作系统，分区 0 为 BOOT 区，存储 BOOTLOAD，如 BIOS，分区 1 用做存储内核，用户也可将自己的程序烧入此分区，分区 2 用作存储根文件系统。BIOS 在启动后，若等待 5S 钟没接收到控制串口的数据，会自动将存储在分区 1 的程序读到 0x30400000 地址开始的 SDRAM 中并运行。用户可在下载完数据后不运行，输入 2 后选择分区将下载的数据烧写到 NAND FLASH 里去，注意若下载的文件大于分区大小时，BIOS 会有提示，但仍可执行烧写。

功能[0]：通过 USB DEVICE 下载程序，并运行。

功能[1]：通过 USB DEVICE 下载程序，不立即运行，下载完后，会自动返回到主功能菜单。在选择这个功能选项后，要输入一个下载地址，测试程序的运行地址为 0x30100000,不要与测试程序的地址相冲突就行。

功能[2]：通过串口下载程序

功能[3]：向 NOR FLASH 写入文件或程序

功能[4]：向 NAND FLASH 写文件或程序，NAND FLASH 分为 4 个分区，分区 1 为 BOOT 分区，分区 2 为内核分区（Linux 内核），分区 2 根文件系统分区，分区 3 其

他文件系统分区(出厂设置为 WinCE 内核的分区)。

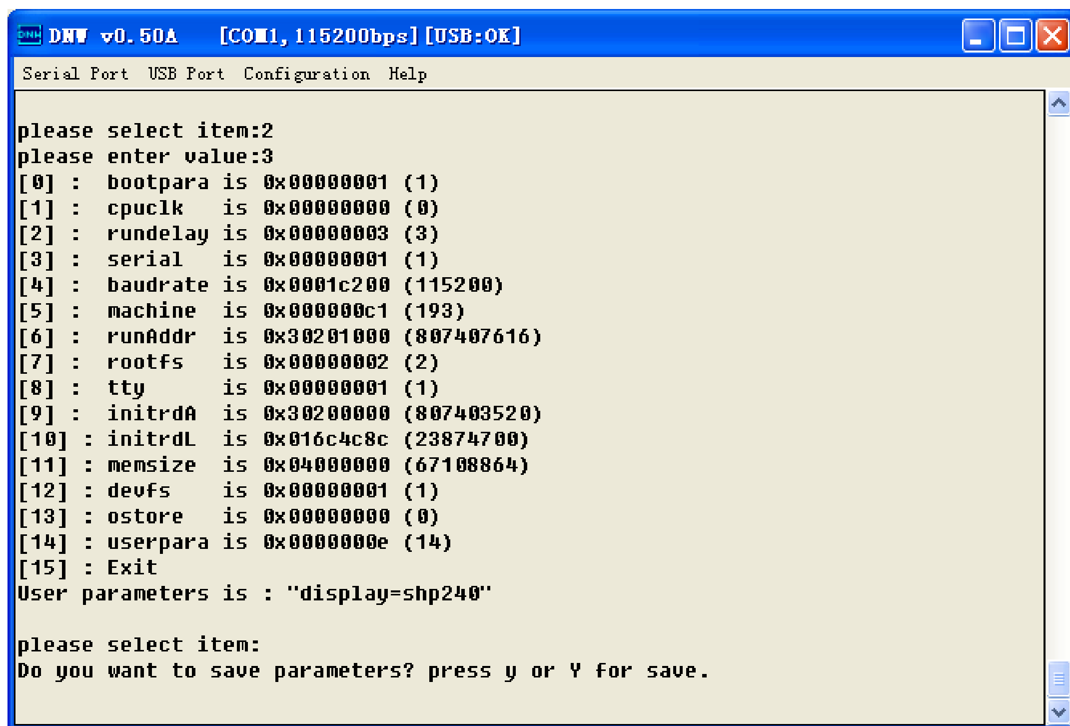
功能[5]：从 NAND FLASH 运行程序。

功能[6]：用来擦除 NAND FLASH 分区

功能[7]：配置一些系统参数和 Linux 启动参数。

功能[8]：解压缩 WINCE 的 Bin 文件

在**功能[7]**中，会出现 14 个配置，下面主要介绍几个关键的参数。设置方法是输入选项编号，如在提示"please select item:："提示下输入 14，出现"please enter value："提示，输入"display=shp240"后回车，再次出现菜单，要保存参数的话，选择 **15** 退出，退出时出现提示是否需要保存，按"Y"后系统会保存参数退出，显示主菜单。具体操作可以参看下图：



参数[0]：启动分区选择，1 为 Linux 系统，3 为 WINCE 系统

参数[2]：rundelay ,这个选项用来设置自启动延迟的时间。设置为 0，则不自启动，设置某个非 0 参数，则 BOOTLOADER 启动后，不按任意键延迟一段时间后，会自启动 NAND FLASH 分区 1 里的程序。

参数[7]：rootfs,设置 Linux 根文件系统所在的分区，1 为 1 分区，2 为 2 分区。一般情况下设为“2”。

参数[14]：userpara,设置 LCD 显示参数，对于 Sharp 3.5”（320X240）,设置的参数为 display=shp240。参数设置后，选择 15 退出，将会提示是否要固化，输入“y”，这样就将参数固化到 FLASH 中了。

3.3.2. 非操作系统下的外围资源测试

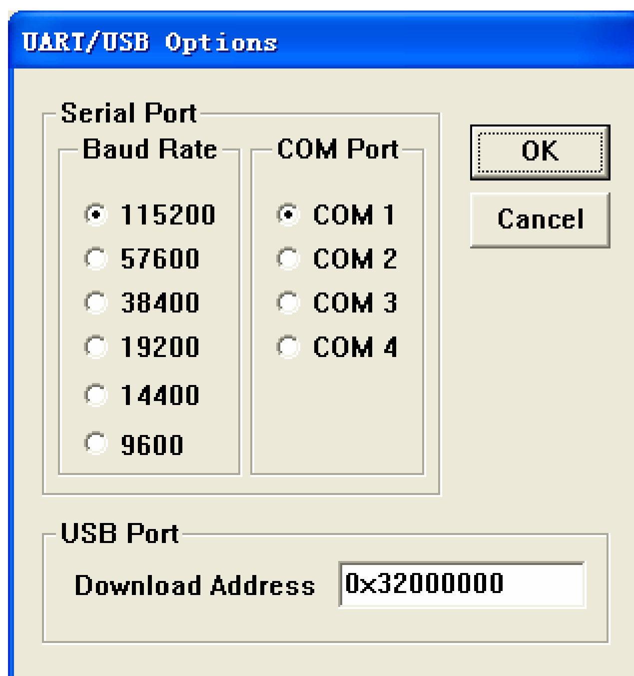
在非操作系统下，主要测试蜂鸣器测试，RTC 实时时钟测试，ADC 测试，按键测试，触摸屏测试，LCD 屏测试，640x480 VGA 测试，红外线测试，CAN 总线测试，IIC 测试，音频输出测试，音频输入测试， SD 卡测试，CF 卡测试以及摄像头测试。

3.3.2.1. 运行测试程序

先找到 BIOS 测试程序 YL2440A_Test.bin，这个 bin 文件在光盘目录的 “目标代码” 文件夹下）下载运行，首先下载测试程序，下载步骤如下：

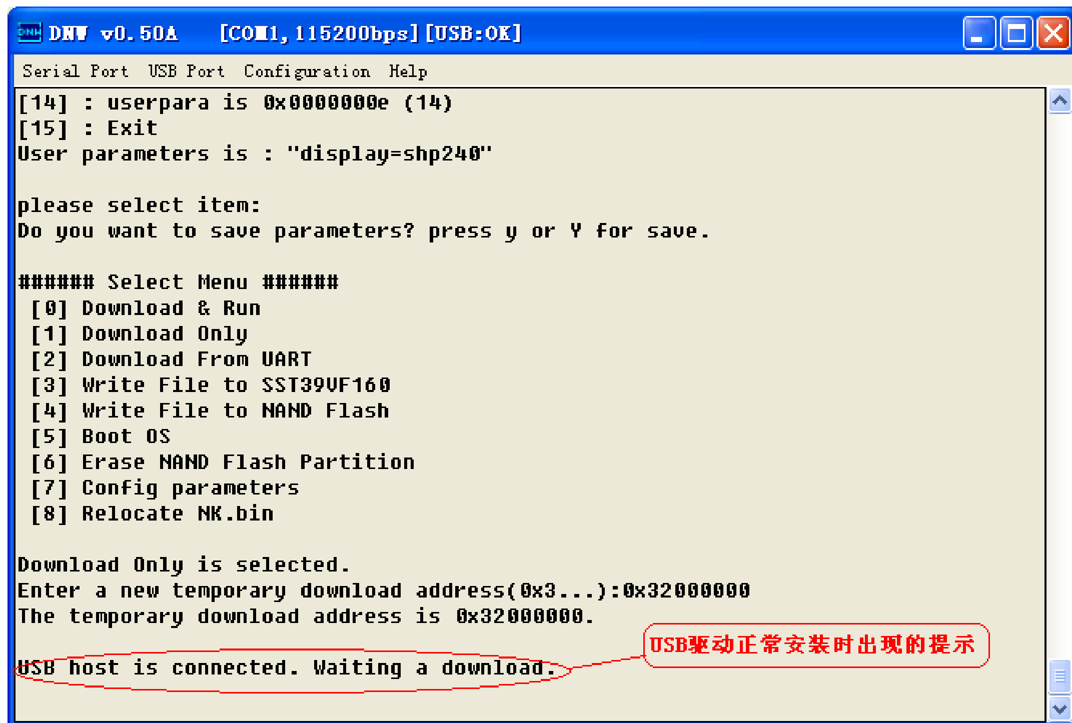
（1） 接好开发板电源，上电启动开发板，按回车键，进入 BOOTLOADER 的主功能菜单。

（2） 在 DNW 中设置 USB 下载运行地址为 0x32000000。

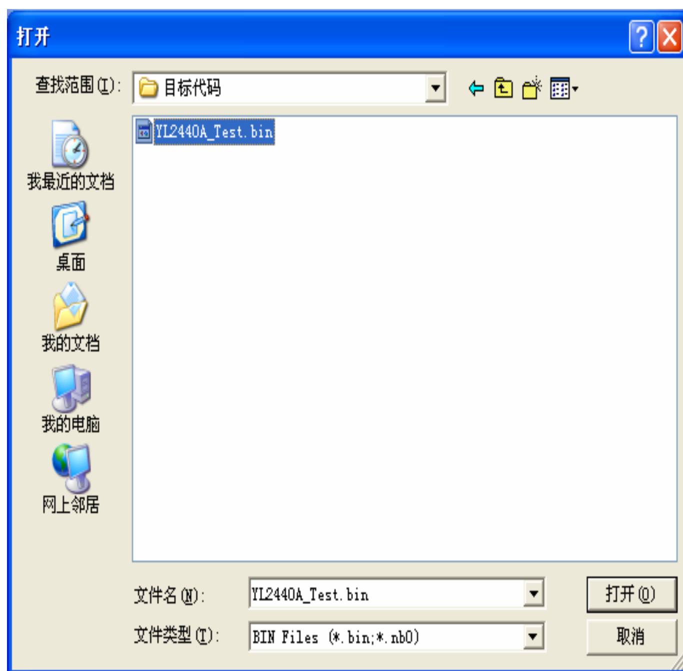


（3） 接上 USB DEVICE 的连接线，这时要保证 USB 驱动程序已装好（前面已详细讲过 USB 驱动的具体安装），在 DNW 中输入“0”进入 USB 下载并运行，出现“USB host is connected. Waiting a download.”说明 USB 正确枚举了，这时可以通过 USB

下载了。（注意用 USB 下载要先在 PC 端装好驱动程序，保证 USB 连接好，有时 PC 端出现发现无法识别的 USB 设备时，可在 BIOS 中输入 ESC 取消下载，等几秒钟再输入“0”启动 USB 下载。USB 连接是否成功在启动信息中有提示, 如下图）

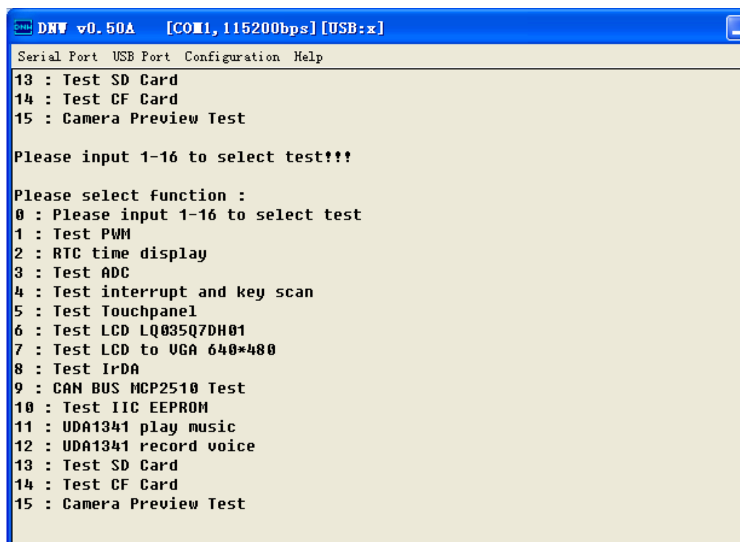


（4） 点击“USB Port”→”Transmit”选项，选择 YL2440A_Test.bin 这个映像文件在光盘目录的“目标代码”文件夹下，



接着点击打开，这样就开始下载了。

（4） 下载结束后，会自动运行。 出现如下界面：



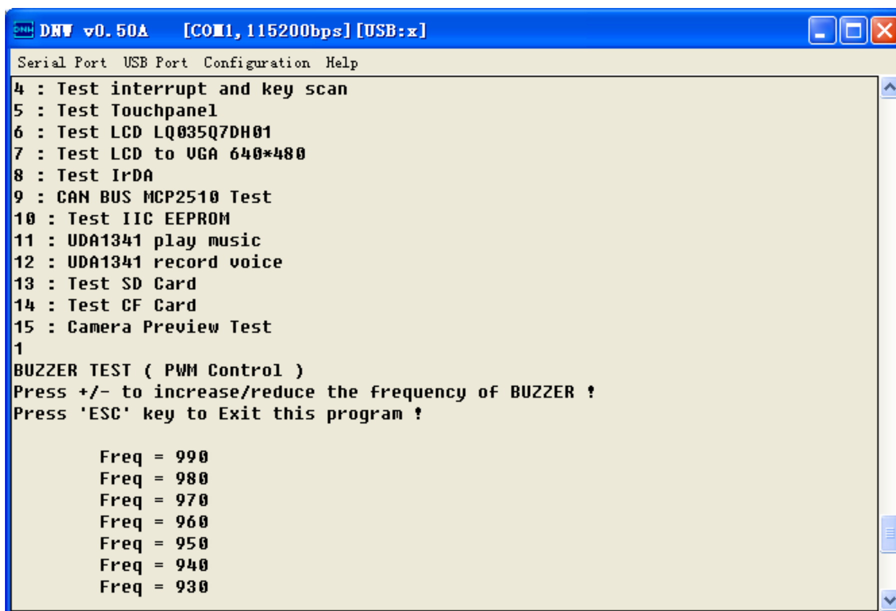
3.3.2.2.相关外围资源测试

测试程序运行后，就可以进行相应的外围资源测试了，通过选择测试程序主功能菜单相应的选项，就可以进行测试。

（1）蜂鸣器测试（Test PWM）

在主菜单中，输入“1”，再按“回车”键（即 Enter 键），将进行蜂鸣器测试，蜂鸣器

测试运行起来，将会听到蜂鸣器发出叫声。



按“－”键蜂鸣器频率会降低，按“＋”键频率升高，按“ESC”键可以退出该测试，并返回到主菜单中。

（2）实时时钟测试

在测试程序主菜单中，选择“2”，再按“回车”键（即 Enter 键），

看到秒钟在不断的变化，说明 RTC 在正常工作，有一点要说明的是，这个时间并不是当前的时间。

```
DNW v0.50A    [COM1,115200bps][USB:x]
Serial Port  USB Port  Configuration  Help
1 : Test PWM
2 : RTC time display
3 : Test ADC
4 : Test interrupt and key scan
5 : Test Touchpanel
6 : Test LCD LQ035Q7DH01
7 : Test LCD to VGA 640*480
8 : Test IrDA
9 : CAN BUS MCP2510 Test
10 : Test IIC EEPROM
11 : UDA1341 play music
12 : UDA1341 record voice
13 : Test SD Card
14 : Test CF Card
15 : Camera Preview Test
2RTC TIME Display, press ESC key to exit !
RTC time : 2005-06-19 15:21:30
RTC time : 2005-06-19 15:21:31
RTC time : 2005-06-19 15:21:31
RTC time : 2005-06-19 15:21:32
RTC time : 2005-06-19 15:21:33
RTC time : 2005-06-19 15:21:34
RTC time : 2005-06-19 15:21:35
RTC time : 2005-06-19 15:21:36
```

按“ESC”键可以退出该测试，并返回到主菜单中。

（3）ADC 测试

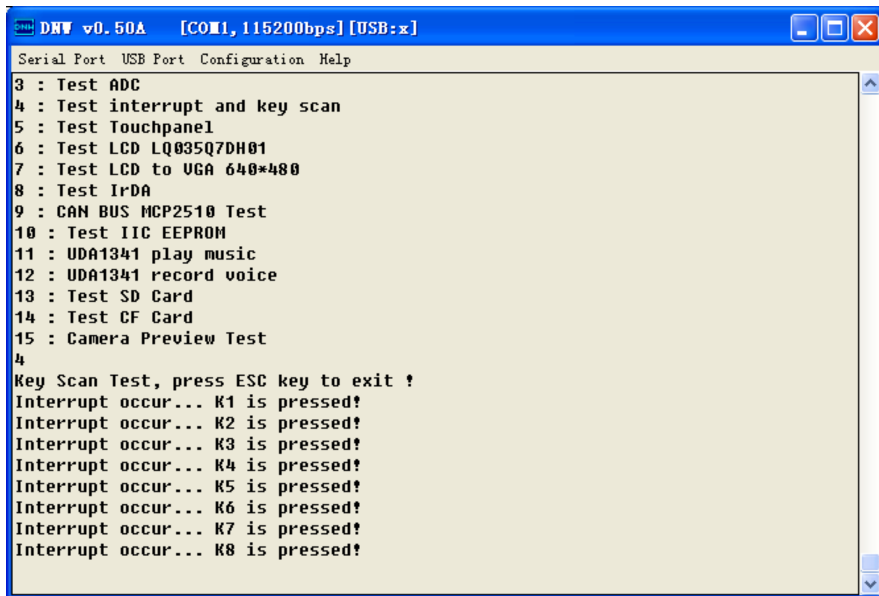
在主菜单中，输入“3”，再按“回车”键（即 Enter 键），将进行 ADC 测试，可以用螺丝刀调节开发板右下角的 R107（AIN0），可以看到 ADC 的值在不断的变化。

```
DNW v0.50A    [COM1,115200bps][USB:x]
Serial Port  USB Port  Configuration  Help
4 : Test interrupt and key scan
5 : Test Touchpanel
6 : Test LCD LQ035Q7DH01
7 : Test LCD to VGA 640*480
8 : Test IrDA
9 : CAN BUS MCP2510 Test
10 : Test IIC EEPROM
11 : UDA1341 play music
12 : UDA1341 record voice
13 : Test SD Card
14 : Test CF Card
15 : Camera Preview Test
3ADC INPUT Test, press ESC key to exit !
ADC conv. freq. = 2500000Hz
PCLK/ADC_FREQ - 1 = 19
AIN0: 0412
AIN0: 0373
AIN0: 0338
AIN0: 0340
AIN0: 0348
AIN0: 0354
AIN0: 0344
AIN0: 0343
AIN0: 0357
```

按“ESC”键可以退出该测试，并返回到主菜单中

（4）按键测试

在主菜单中，输入“4”，再按“回车”键（即 Enter 键），将进行按键测试，按键测试运行起来后，可以按开发板上的 S1~S8 按键进行测试，按一个键将在串口打印相应的按键信息。

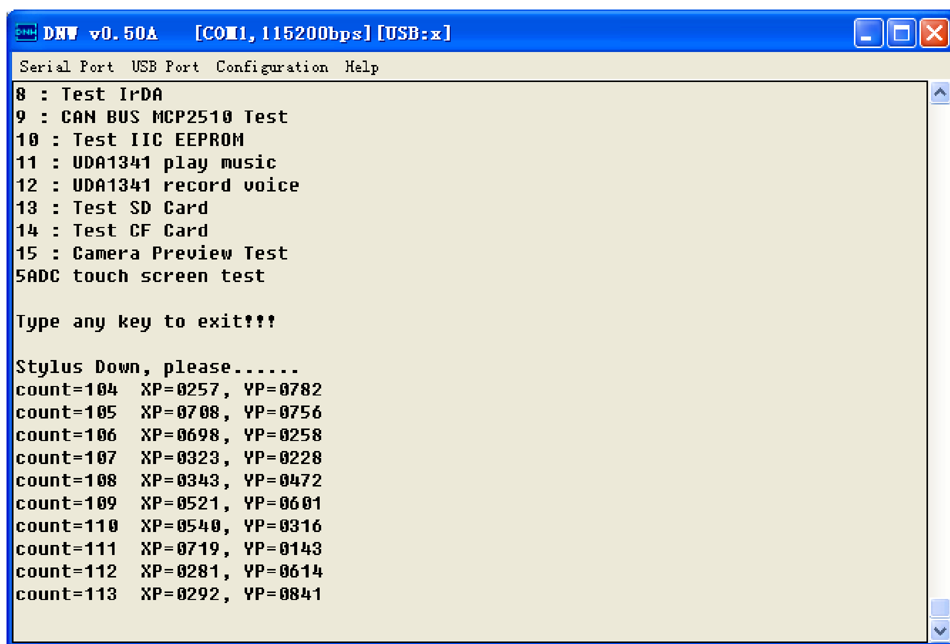


按“ESC”键可以退出该测试，并返回到主菜单中。

（5）触摸屏测试

如果选购了 LCD 屏，请用 50 针的排线，将 LCD 屏与开发板的 LCD 接口连接起来。

在主菜单中，输入“5”，再按“回车”键（即 Enter 键），将进行触摸屏测试，接着点击触摸屏，可以看到串口可以打印出触点的位置。



按“ESC”键可以退出该测试，并返回到主菜单中。

（6）LCD 测试

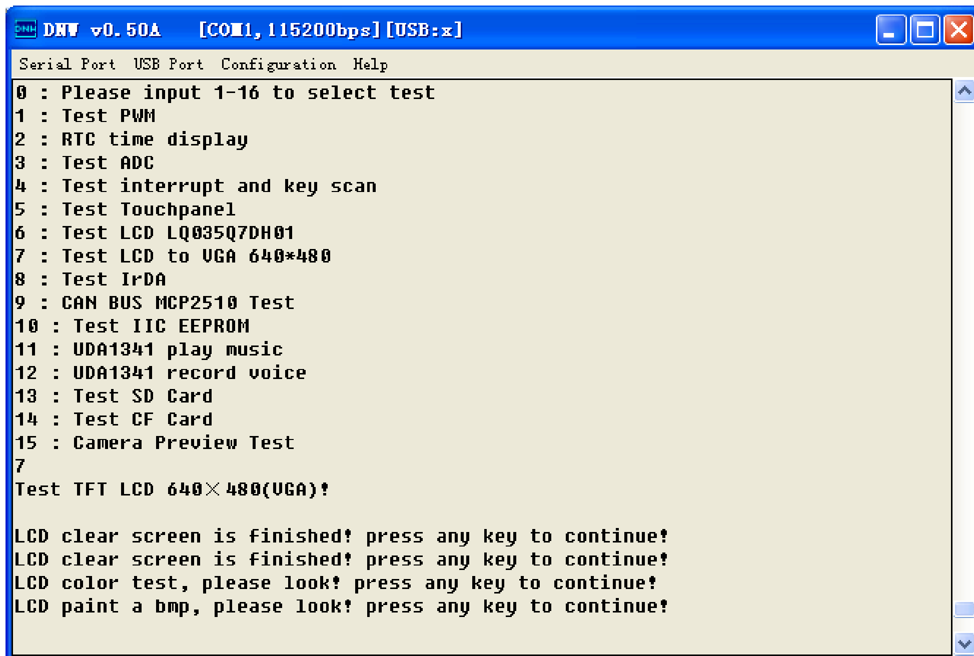
如果选购了 LCD 屏，请用 50 针的排线，将 LCD 屏与开发板的 LCD 接口连接起来。

在主菜单中，输入“6”，再按“回车”键（即 Enter 键），将进行 LCD 测试，按任意键将显示继续，接着再按“回车”键（即 Enter 键），直到图片都显示结束，将退出 LCD 测试，返回主功能菜单。



（7）640x480 VGA 测试

在主菜单中，输入“7”，再按“回车”键（即 Enter 键），将进行 640x480 VGA 测试，接着将 YL2440 的 VGA 接口与 VGA 彩显相连。



按任意键将显示继续，接着再按“回车”键（即 Enter 键），直到图片都显示结束，将退出 640x480 VGA 测试，返回主功能菜单。

（8）红外线测试

在主菜单中，输入“8”，再按“回车”键（即 Enter 键），将进行红外线测试，红外线进行的是自收发模式测试，可以看到发一个立刻接收一个数据,如下图：

DNW v0.50A [COM1,115200bps][USB:x]
Serial Port USB Port Configuration Help

```
Please select function :
0 : Please input 1-16 to select test
1 : Test PWM
2 : RTC time display
3 : Test ADC
4 : Test interrupt and key scan
5 : Test Touchpanel
6 : Test LCD LQ035Q7DH01
7 : Test LCD to VGA 640*480
8 : Test IrDA
9 : CAN BUS MCP2510 Test
10 : Test IIC EEPROM
11 : UDA1341 play music
12 : UDA1341 record voice
13 : Test SD Card
14 : Test CF Card
15 : Camera Preview Test
8
IrDA test
Select baudrate = 19200
Now start UART IrDA Tx&Rx test, check data[a~z]
aabbccddeeffgghhiijjkkllmmnnooppqqrrssttuuvvwwxxyyz
End Tx&Rx
```

测试结束后，会自动返回到主菜单界面。

（9）CAN 总线测试

在主菜单中，输入“9”，再按“回车”键（即 Enter 键），将进行 CAN 总线测试，我们让 CAN 工作在回环模式，可以在串口的显示信息中看到 CAN 通讯的结果。

```
DNW v0.50A    [COM1,115200bps][USB:x]
Serial Port  USB Port  Configuration  Help
5 : Test Touchpanel
6 : Test LCD LQ035Q7DH01
7 : Test LCD to VGA 640*480
8 : Test IrDA
9 : CAN BUS MCP2510 Test
10 : Test IIC EEPROM
11 : UDA1341 play music
12 : UDA1341 record voice
13 : Test SD Card
14 : Test CF Card
15 : Camera Preview Test
9
CAN BUS Test[ MCP2510 ], press ESC key to exit !
MCP2510REG_CANCTRL =  0x80
  ID=0x5a5
Data=1,2,3,4,5,6,7,8
  ID=0x5a5
Data=1,2,3,4,5,6,7,8
  ID=0x5a5
Data=1,2,3,4,5,6,7,8
  ID=0x5a5
Data=1,2,3,4,5,6,7,8
  ID=0x5a5
Data=1,2,3,4,5,6,7,8
```

按“ESC”键可以退出该测试，并返回到主菜单中。

（10）IIC 测试

在主菜单中，输入“10”，再按“回车”键（即 Enter 键），将进行 IIC 读写测试，这个测试，主要是通过向 AT24C02 写 0~255 的数据，然后读出来。

```
DNW v0.50A   [COM1,115200bps][USB:x]
Serial Port  USB Port  Configuration  Help
13 : Test SD Card
14 : Test CF Card
15 : Camera Preview Test
10
IIC Test(Interrupt) using AT24C02
Write test data into AT24C02
Read test data from AT24C02
 0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f
20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f
30 31 32 33 34 35 36 37 38 39 3a 3b 3c 3d 3e 3f
40 41 42 43 44 45 46 47 48 49 4a 4b 4c 4d 4e 4f
50 51 52 53 54 55 56 57 58 59 5a 5b 5c 5d 5e 5f
60 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f
70 71 72 73 74 75 76 77 78 79 7a 7b 7c 7d 7e 7f
80 81 82 83 84 85 86 87 88 89 8a 8b 8c 8d 8e 8f
90 91 92 93 94 95 96 97 98 99 9a 9b 9c 9d 9e 9f
a0 a1 a2 a3 a4 a5 a6 a7 a8 a9 aa ab ac ad ae af
b0 b1 b2 b3 b4 b5 b6 b7 b8 b9 ba bb bc bd be bf
c0 c1 c2 c3 c4 c5 c6 c7 c8 c9 ca cb cc cd ce cf
d0 d1 d2 d3 d4 d5 d6 d7 d8 d9 da db dc dd de df
e0 e1 e2 e3 e4 e5 e6 e7 e8 e9 ea eb ec ed ee ef
f0 f1 f2 f3 f4 f5 f6 f7 f8 f9 fa fb fc fd fe ff
```

这个测试结束后，会自动退回到主菜单中。

（11）音频输出测试

先将音箱接到开发板的 J5 接口。

在主菜单中，输入“11”，再按“回车”键（即 Enter 键），将进行音频输出测试，这时

将从音箱听到声音。

```
DNW v0.50A  [COM1,115200bps][USB:x]
Serial Port  USB Port  Configuration  Help
3 : Test ADC
4 : Test interrupt and key scan
5 : Test Touchpanel
6 : Test LCD LQ035Q7DH01
7 : Test LCD to VGA 640*480
8 : Test IrDA
9 : CAN BUS MCP2510 Test
10 : Test IIC EEPROM
11 : UDA1341 play music
12 : UDA1341 record voice
13 : Test SD Card
14 : Test CF Card
15 : Camera Preview Test
11
Sample Rate = 22050, Channels = 2, 16BitsPerSample, size = 243508

err = 0
Now playing the file
Press 'ESC' to quit, '+' to inc volume, '-' to dec volume, 'm' to mute, 'p' to
pause

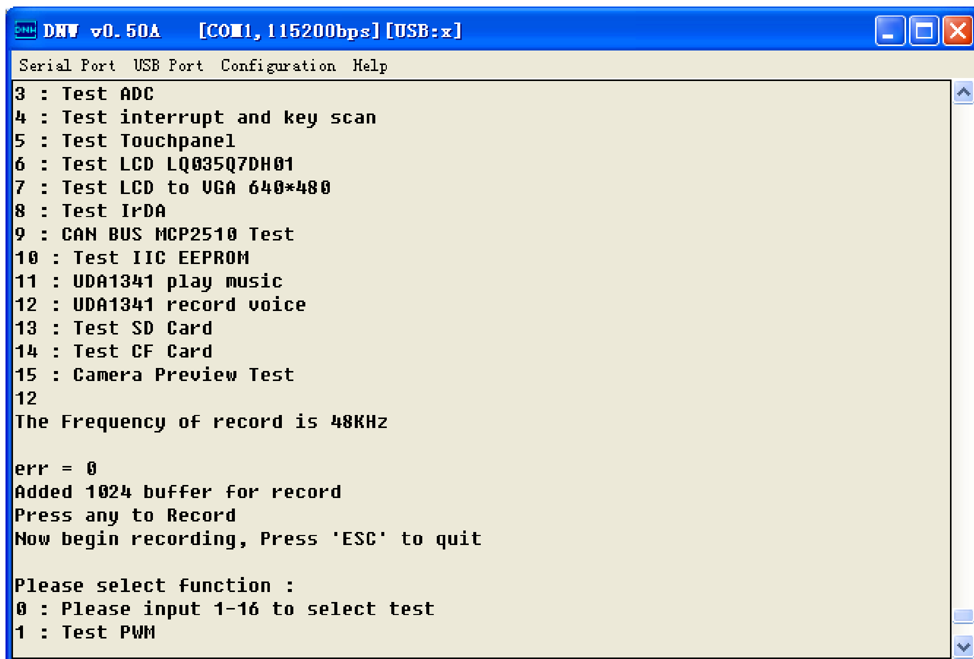
Please select function :
0 : Please input 1-16 to select test
1 : Test PWM
```

按“+”或“-”可以增加或减少音量，按“ESC”键可以退出测试，返回主菜单。

（12）音频输入测试

先将音箱接到开发板的 J5 接口，并将麦克风与开发板的 J4 相连。

在主菜单中，输入“12”，再按“回车”键（即 Enter 键），将进行音频输入测试，这时按照该测试的提示，输入任意键，将会进行录音测试，可以对着开发板的 MIC（MK1）或麦克风发声，在音箱处将会听到声音。



按“ESC”键可以退出该测试，并返回到主菜单中。

（13）SD 卡测试

先将 SD 卡插入开发板的 SD 卡座。

在主菜单中，输入“13”，再按“回车”键（即 Enter 键），将进行 SD 卡读写测试，出现下图的界面。

```
DNW v0.50A    [COM4,115200bps][USB:x]
Serial Port  USB Port  Configuration  Help
15 : Test SD Card
16 : Camera Preview Test
15
SDI Card Write and Read Test
Init. Frequency is 301204Hz
In idle
MMC check end!!
In SD ready
End id
RCA=0xe624
SD Frequency is 25000000Hz
In stand-by
End Rx buffer flush
Block write test[ Polling write ]
Block read test[ Polling read ]
Check Rx data

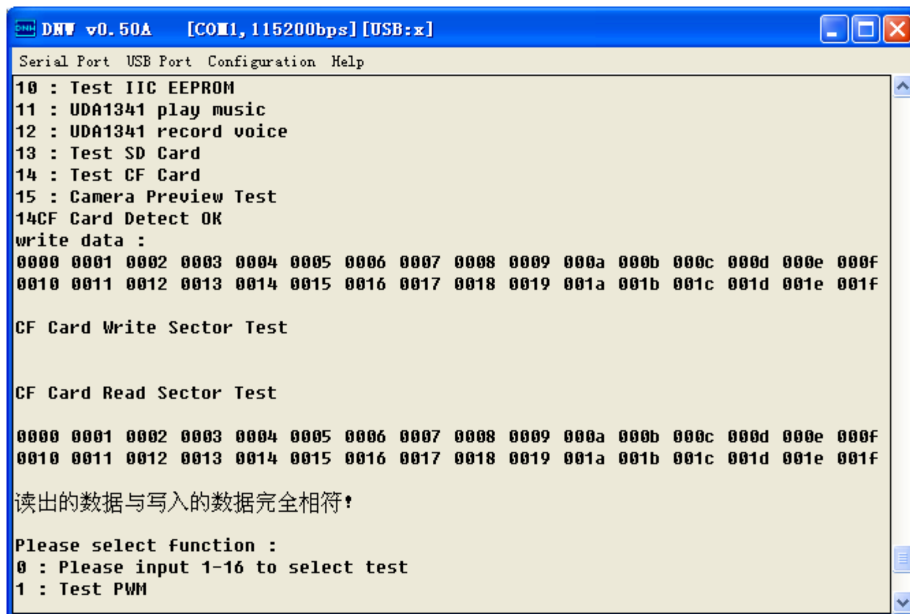
The Tx_buffer is same to Rx_buffer!
SD CARD Write and Read test is OK!

CSD register :
SDIRSP0=0x260032
SDIRSP1=0x1f5981d2
SDIRSP2=0xfef9cfff
SDIRSP3=0x92404083
```

出现上面红框的提示，说明 SD 测试是 OK 的，测试完成后，会自动退回到主菜单。

（14）CF 卡读写测试

先插上 CF 卡，再上电运行测试程序，进入测试主菜单，输入“14”，在按“回车”（即 Enter 键），将进行 CF 卡读写测试，测试运行后，将在串口打印写入 CF 卡的数据和从 CF 读出的数据，对这写入的数据和读出的数据进行比较，能验证 CF 的接口的好坏。



测试完成后，会自动退回到主菜单。

（15）摄像头测试

如果选购了 LCD 屏，请用 50 针的排线，将 LCD 屏与开发板的 LCD 接口连接起来，并确保摄像头已装在开发板上。

在主菜单中，输入“15”，再按“回车”键（即 Enter 键），将进行摄像头测试，在 DNW 中显示如下图示：

```
DNW v0.50A [COM1,115200bps][USB:x]
Serial Port  USB Port  Configuration  Help
3 : Test ADC
4 : Test interrupt and key scan
5 : Test Touchpanel
6 : Test LCD LQ035Q7DH01
7 : Test LCD to VGA 640*480
8 : Test IrDA
9 : CAN BUS MCP2510 Test
10 : Test IIC EEPROM
11 : UDA1341 play music
12 : UDA1341 record voice
13 : Test SD Card
14 : Test CF Card
15 : Camera Preview Test
15
Camera Preview Test
CAMERA : UPLL 96000000  UCLK 48000000 CAMCLK 24000000
Check camera ID
Initial Camera now, Please wait several minutes...
Initializing end...

Now Start Camera Preview
preview sc control = 0
preview sc control = 81550155
Press 'ESC' key to exit!
```

这时将在 LCD 屏的上半部显示摄像头摄到了图象。显示屏显示如下图所示：



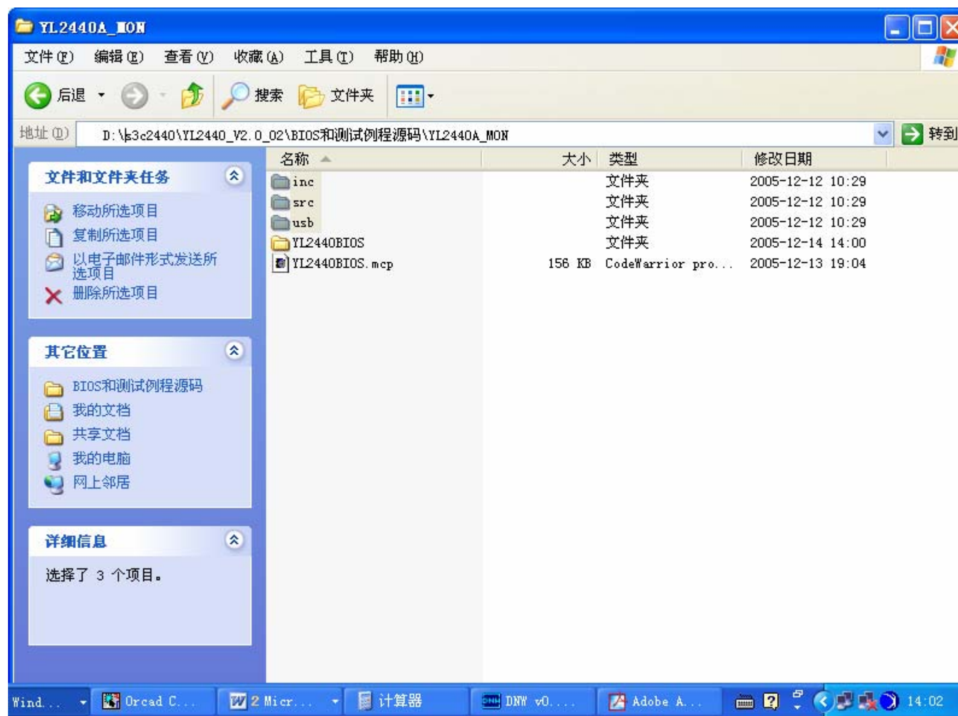
按“ESC”键可以退出该测试，并返回到主菜单中。

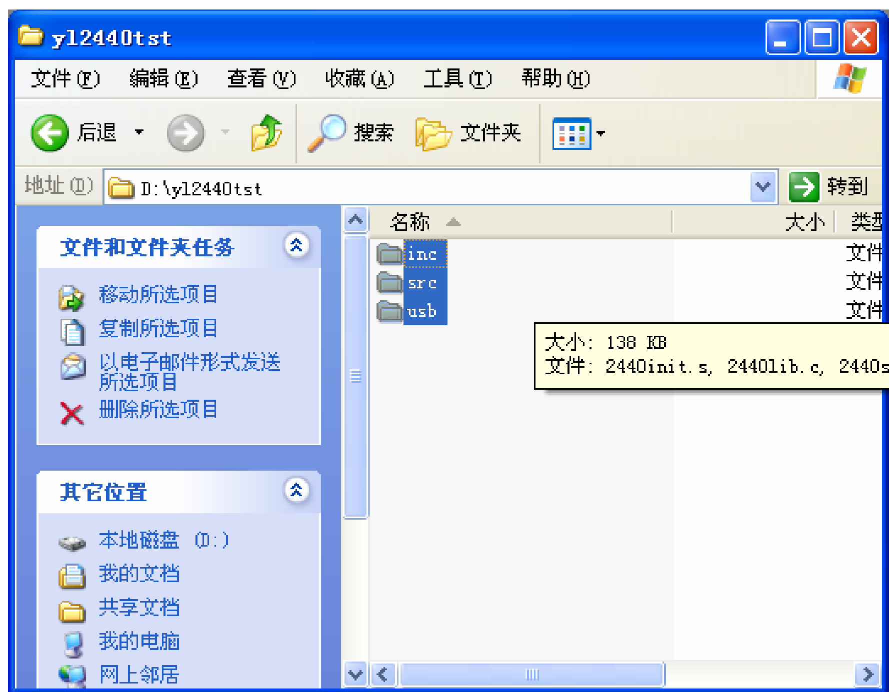
3.4.BIOS 编译测试

在本开发板配套光盘中我们提供了 BIOS 源码，可以将其拷贝到自定的目录下对其进行编译测试。

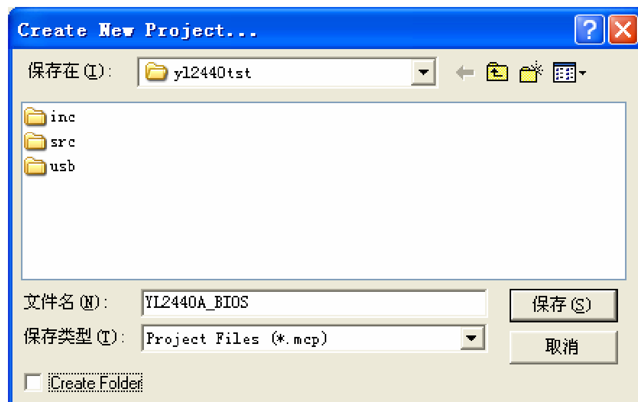
操作步骤如下：

１．把光盘中的“BIOS 和测试例程源码”文件夹打开，进入“YL2440A_MON”文件夹，将如图三个文件夹复制，拷贝到自定的一个文件夹中，如下示例：

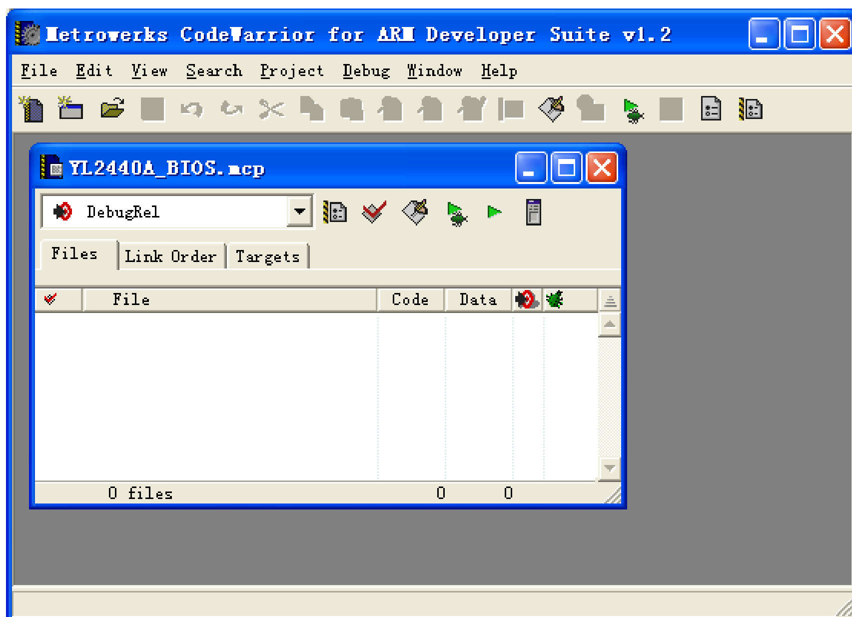

YL2440A_MON
文件(F) 编辑(E) 查看(V) 收藏(A) 工具(T) 帮助(H)
后退 搜索 文件夹
地址(D) D:\s3c2440\YL2440_V2.0_02\BIOS和测试例程源码\YL2440A_MON
转到
文件和文件夹任务
移动所选项目
复制所选项目
以电子邮件形式发送所选项目
删除所选项目
其它位置
BIOS和测试例程源码
我的文档
共享文档
我的电脑
网上邻居
详细信息
选择了 3 个项目。
名称 大小 类型 修改日期
inc 文件夹 2005-12-12 10:29
src 文件夹 2005-12-12 10:29
usb 文件夹 2005-12-12 10:29
YL2440BIOS 文件夹 2005-12-14 14:00
YL2440BIOS.mcp 156 KB CodeWarrior pro... 2005-12-13 19:04
Wind...
Orcad C...
2 Micr...
计算器
DNW v0....
Adobe A...
14:02



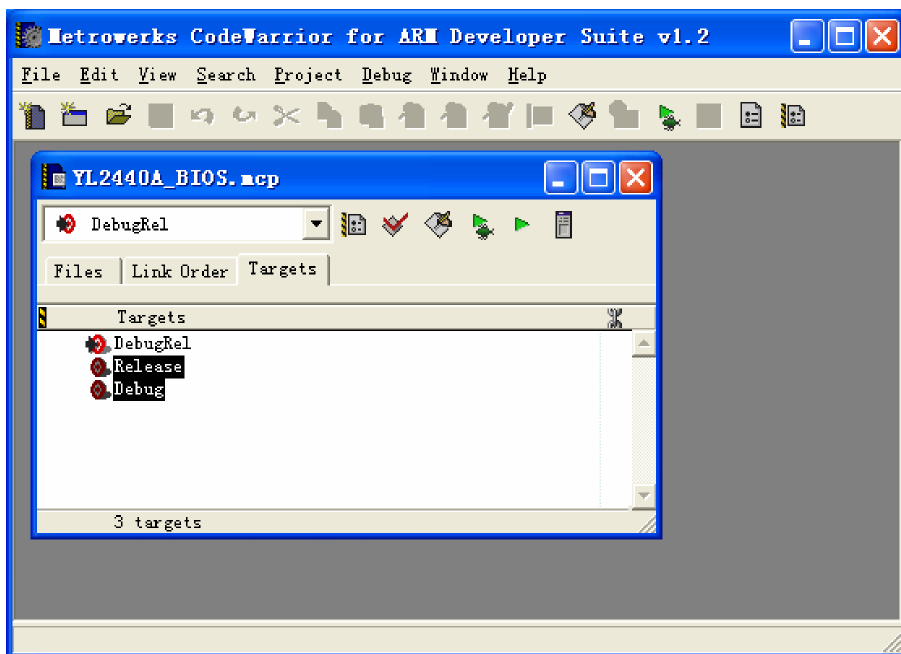
打开 ADS1.2 编译环境，新建一个工程，指定路径在自定目录下，如图路径：



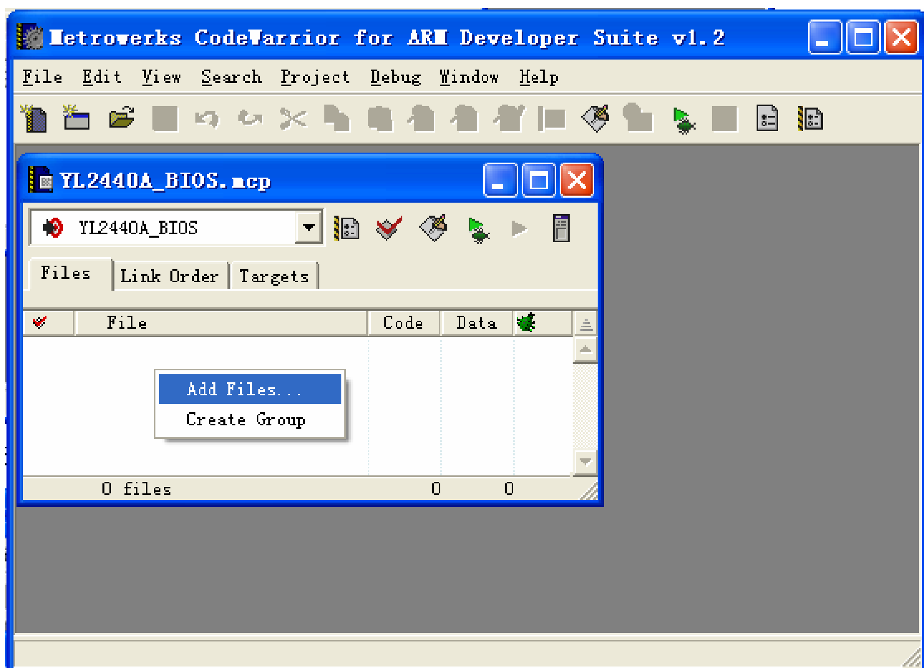
保存完毕，选择“确定”后会新建一个新的工程。如下图：



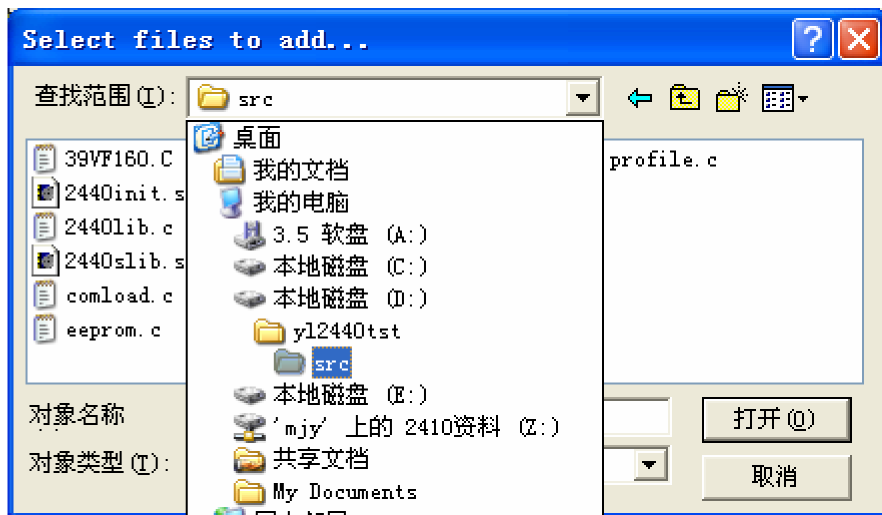
选择 “Targets”，删除掉如下图的目标，



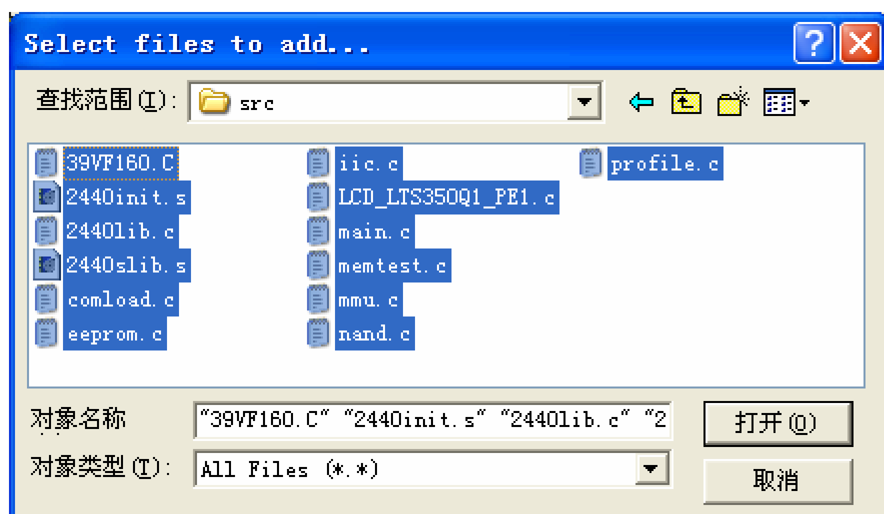
在 F iles 中单击右键，出现“Add Files...”，

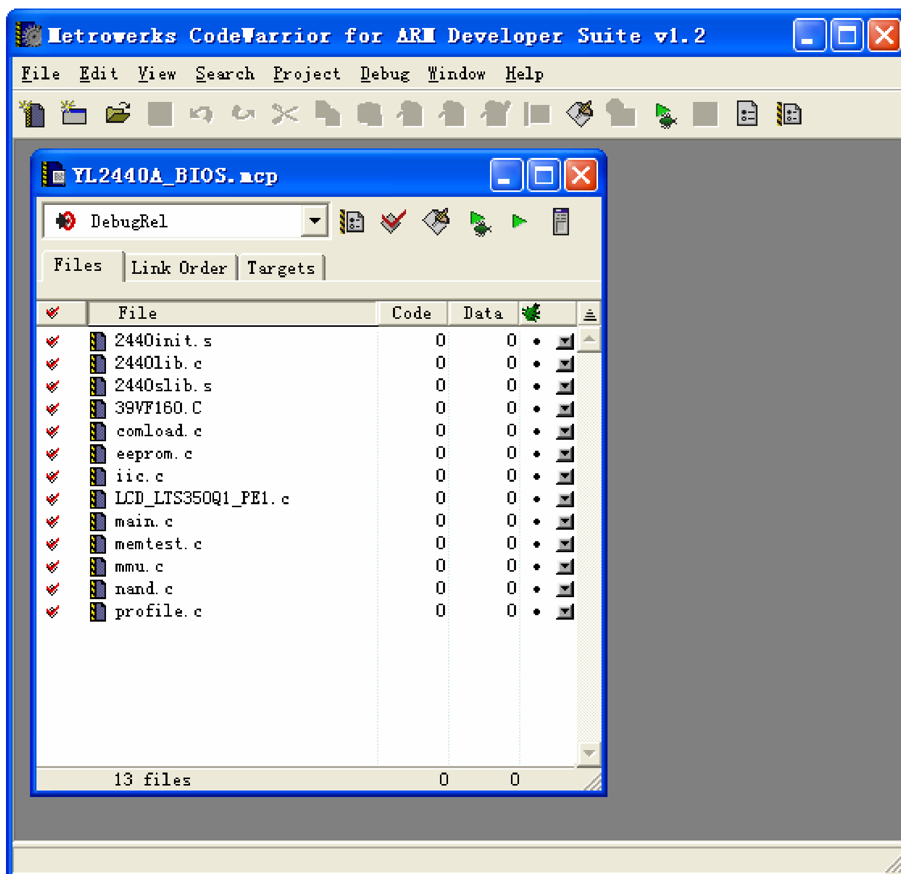


选中后出现添加文件的提示，选定到刚新建工程的文件夹，如下界面：



选中“scr”文件夹中的所有文件，单击“打开”后可以看到文件都添加到工程中，


Select files to add...
查找范围(I):
src
39VF160.C
2440init.s
2440lib.c
2440slib.s
comload.c
eeprom.c
iic.c
LCD_LTS350Q1_PE1.c
main.c
memtest.c
mmu.c
nand.c
profile.c
对象名称
"39VF160.C" "2440init.s" "2440lib.c" "2
对象类型(T):
All Files (*.*)
打开(O)
取消



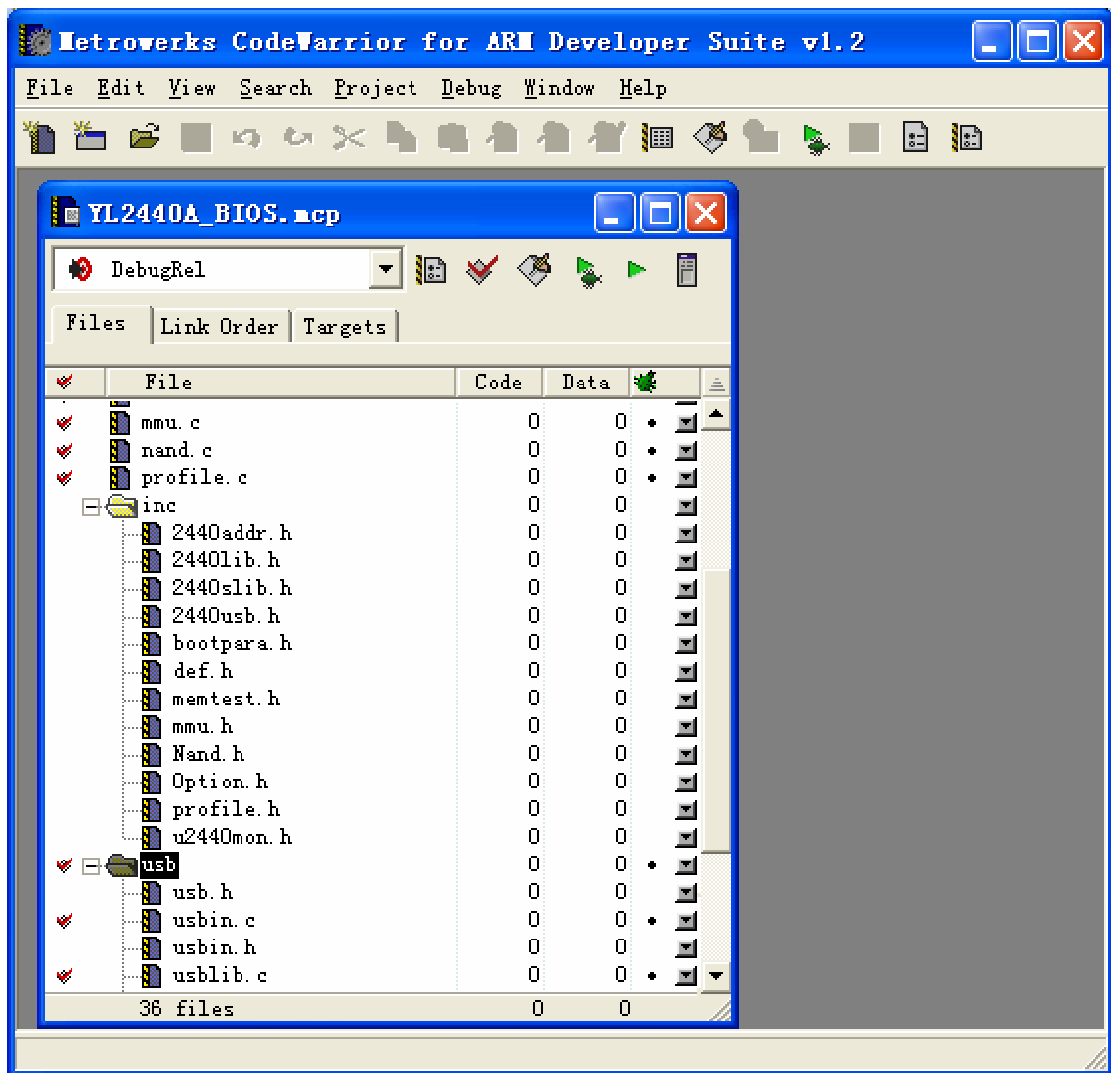
再新建一个文件组，“inc”如下图中，

选择“Creat Group”出现对话框，

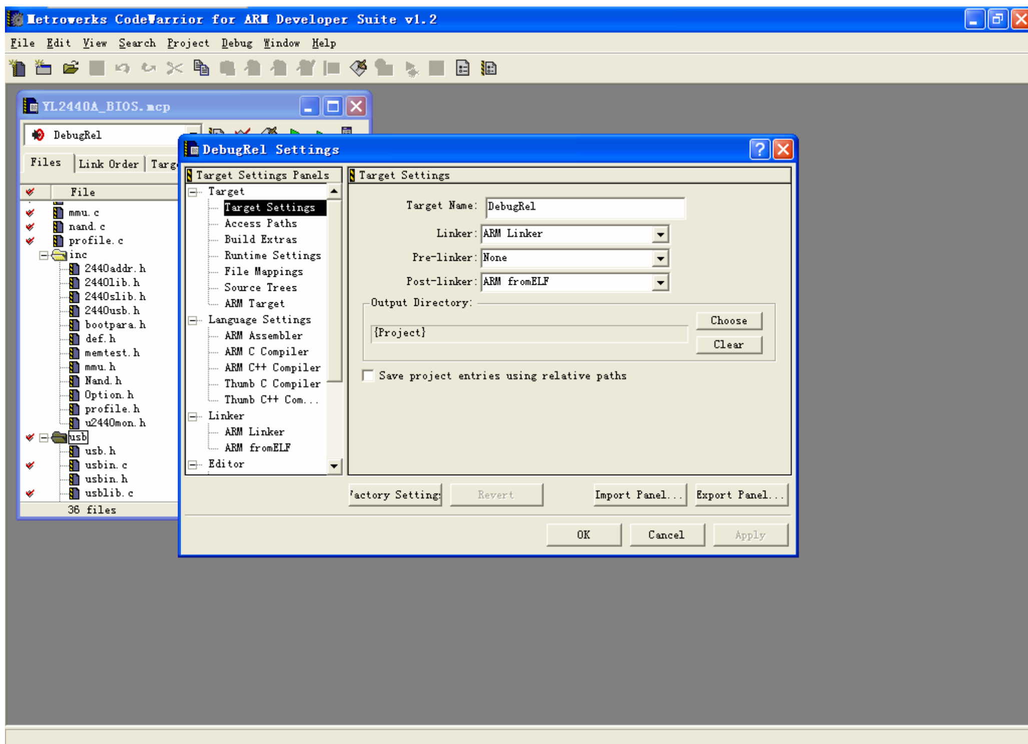


点击“O K”后，再加一个文件组“usb”，

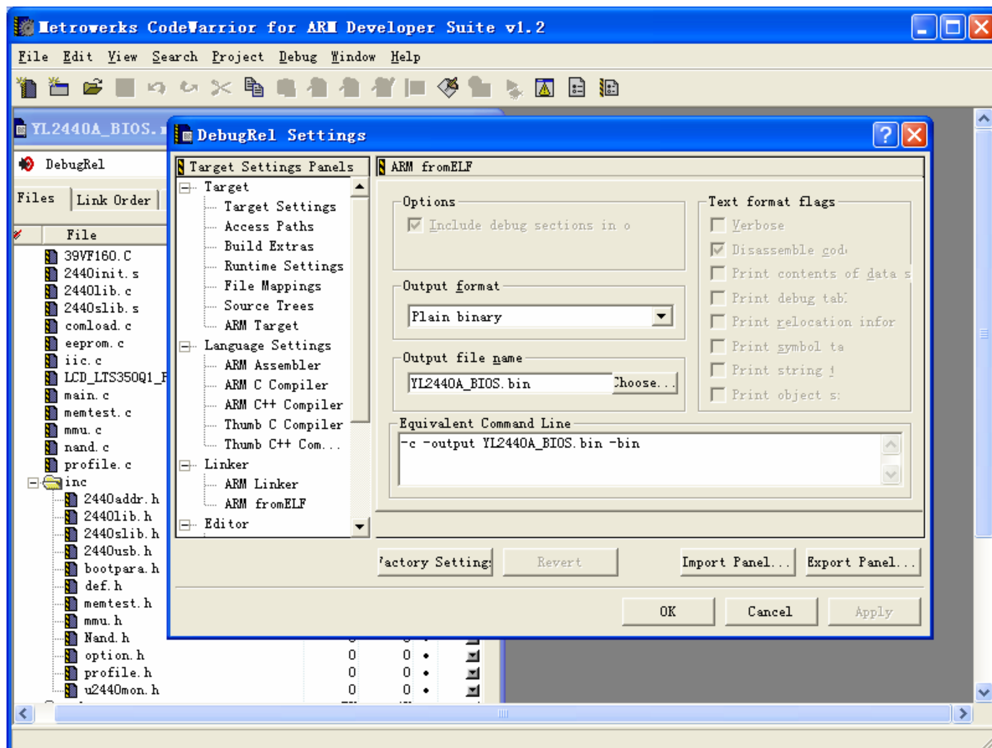
再往文件组中添加文件，将我们拷贝过去的文件夹“inc”中的文件都添加到“inc”组中，“usb”文件组中进行同样的操作，



再打开如图中的“DebugRel setting....”出现下图的界面，设定关键参数。

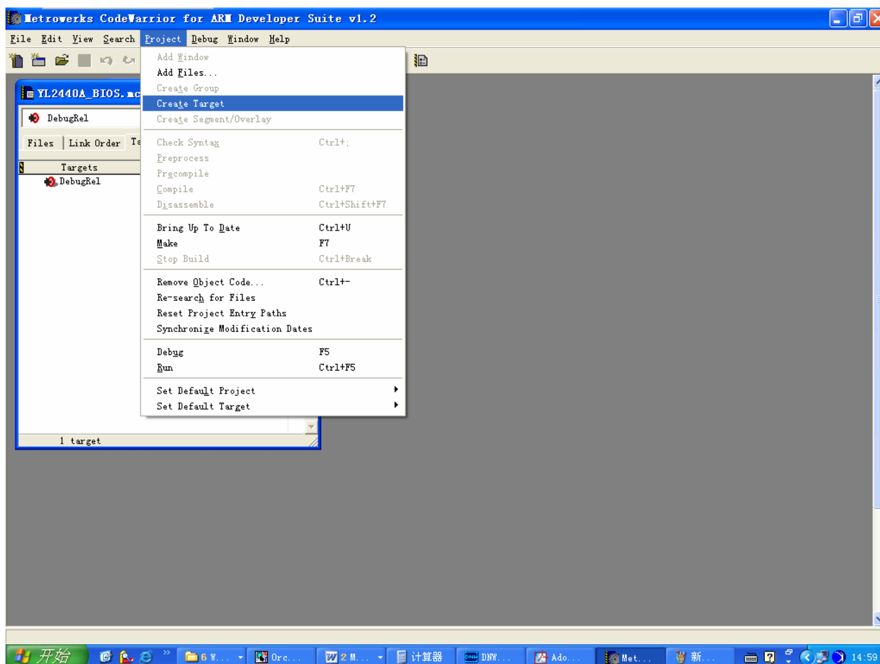


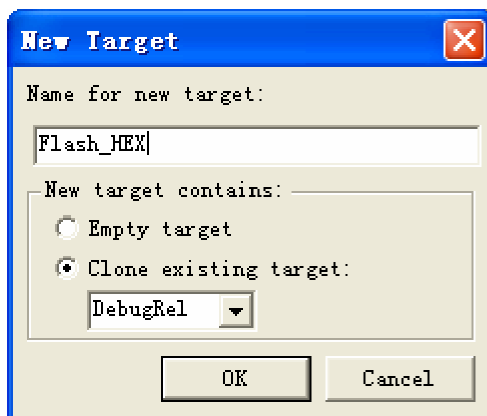
其中生成 bin 文件现在举例如下进行操作：



其中的参数请参照我们给定的工程中的设定进行设置，这里就不多讲述。

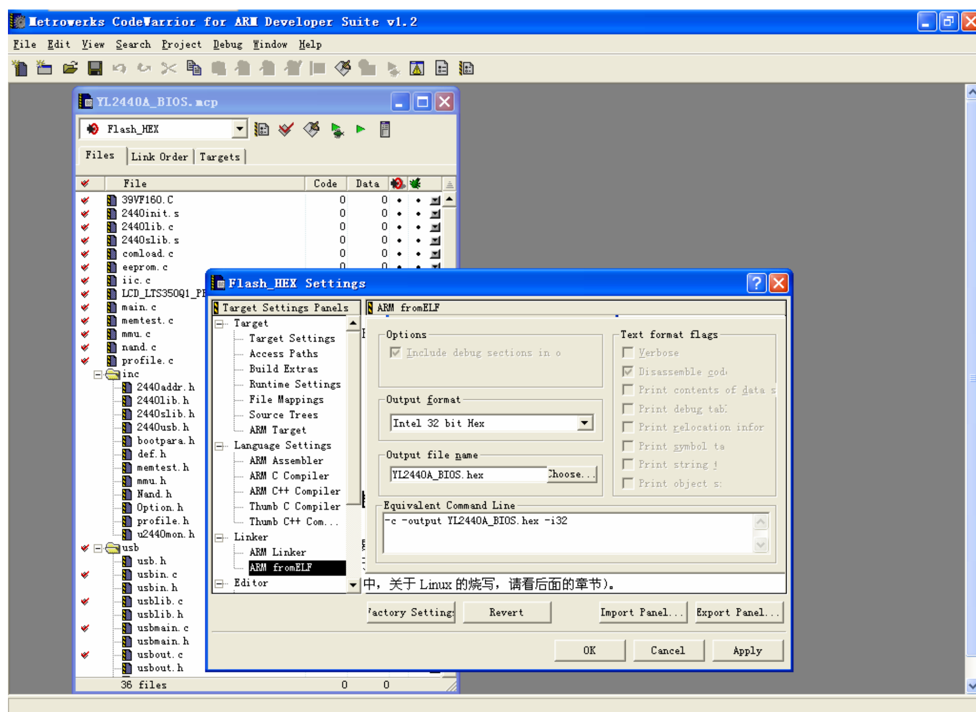
再单击“ T argets”在 Project—> C reat Target





新建一个能生成 HEX 烧写文件的目标文件。

对 F lash_HEX 的参数设定进行配置，需要改动的是如下图



修改好后对工程进行编译。如果出现错误，请仔细参看光盘提供的工程例程。

生成的 bin 文件在当前工程的“YL2440A_BIOS_Data\DebugRel”文件夹下，文件名是自己定义的，接下来进行 BIOS 烧写测试。

3.5. BIOS 烧写测试

我们使用 sjf2440 工具进行烧写，本开发板提供的光盘中有这一烧写工具。

（1） 先用 20 针排线将 YL2440 的 20 针 JTAG 接口（CN5）与 JTAG 小板的 JP3 接口相连。

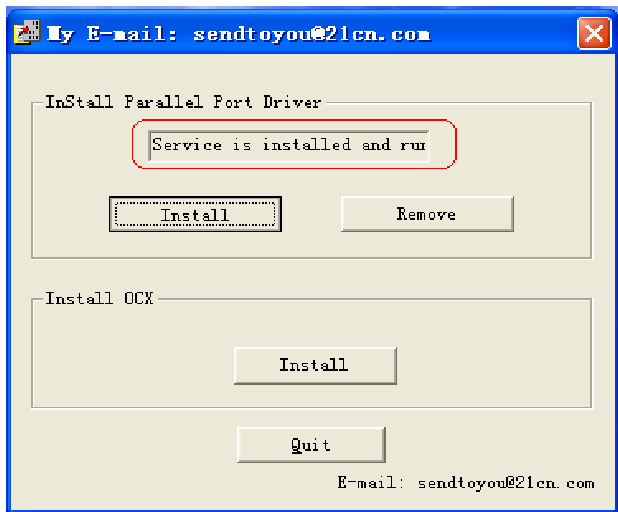
然后将光盘里的“Flash 烧写”文件夹拷贝到 C 盘（硬盘的其他地方也行）。

（2）安装 giveio 驱动，进入“Flash 烧写工具”文件夹下，点击“安装驱动.exe”，将弹

出如下界面。



先点击“InStall Parallel Port Driver”栏目下的“Remove”按纽，然后点击该栏目下的“Install”按纽。



出现“Service is installed and run”说明 giveio 驱动安装成功。

（3）将刚才编译生成的 YL2440A_BIOS.bin 文件复制到当前文件夹下（如果已存在，覆盖它），然后对该文件夹下的 SJF_YL2440A_MON.BAT 批处理文件进行修改，改为“sjf2440

/f:YL2440A_BIOS.bin”。烧写程序所支持的 FLASH 都列出来了，有 K9S1208(NAND，64M)、28F128J3A、AM29LV800、SST39VF160/1 等。

（4） 在出现上面的界面后，我们在“Select the Function to test:”提示下，输入‘0’，然后回车，这时将选择 K9S1208 进行烧写，显示如下：

```
C:\WINDOWS\System32\cmd.exe

C:\Flash烧写>sjf2440 /f:FS2440A_MON.bin

+------------------------------------+
|     SEC JTAG FLASH(SJF) v 0.1      |
|     (S3C2440X & SMDK2440 B/D)      |
+------------------------------------+
Usage: SJF /f:<filename> /d=<delay>
> S3C2440X(ID=0x0032409d) is detected.

[SJF Main Menu]
 0:K9S1208 prog     1:28F128J3A prog   2:AM29LV800 Prog   3:SST39VF1601
 4:Memory Rd/Wr     5:Exit
Select the function to test:0

[K9S1208 NAND Flash JTAG Programmer]
K9S1208 is detected. ID=0xec76
 0:K9S1208 Program      1:K9S1208 Pr BlkPage   2:Exit
Select the function to test :
```

（5）接着在“Select the function to test :”提示下，输入“0”，然后回车，选择 K9F1208 进行烧写，接着再在“Input target block number:”栏下输入偏移地址“0”，显示信息如下：

```
C:\WINDOWS\System32\cmd.exe
Usage: SJF /f:<filename> /d=<delay>
> S3C2440X(ID=0x0032409d) is detected.

[SJF Main Menu]
 0:K9S1208 prog     1:28F128J3A prog    2:AM29LV800 Prog    3:SST39VF1601
 4:Memory Rd/Wr     5:Exit
Select the function to test:0

[K9S1208 NAND Flash JTAG Programmer]
K9S1208 is detected. ID=0xec76
 0:K9S1208 Program       1:K9S1208 Pr BlkPage    2:Exit
Select the function to test :0

[SMC(K9S1208V0M) NAND Flash Writing Program]

Source size:0h~b77bh

Available target block number: 0~4095
Input target block number:0
target start block number      =0
target size          (0x4000*n) =0xc000
STATUS:Epp
```

接着回车，回车后，开始烧写程序，程序烧写完成后，会自动退出。

（6）关闭电源，拔掉 JTAG 插头，将 J8（OM0）的短路帽接上，将 PC 的串口和开发板的串口 2（P3，中间的那个串口）通过串口线连接好，在 PC 上启动 DNW 程序，并通过 Configuration 选项设置好 PC 的串口和波特率，点击 OK 后再在 Serial Port 菜单下选择 Connect，注意不要有其他程序占用你所选中的串口。

打开开发板电源，烧进 NAND FLASH 的程序 BIOS 会启动运行。

3.6. 用 sjf2440 工具将 BIOS 烧写到 Nor Flash

在进行 NOR FLASH 烧写前，先要将 J8（OM0）的短路帽取掉。

（1） 将光盘目录下的“Flash 烧写”文件夹拷贝到 C 盘（也可以是硬盘其他的地方）下。

（2） 安装好 giveio 驱动，如果在上一章节《用 sjf2440 工具将 BIOS 烧写到 NAND FLASH》已安装了，在这一章节，这个步骤可以省略。

（3） 用 20 针排线将 YL2440 的 20 针 JTAG 接口（JTAG）与 JTAG 小板的 JP3 接口相连。

（4） 进入“Flash 烧写”文件夹下，然后点击该文件夹下的 SJF_YL2440A_MON.BAT 批处理文件，显示信息如下：

```
C:\WINDOWS\System32\cmd.exe

C:\Flash烧写>sjf2440 /f:FS2440A_MON.bin

+------------------------------------+
|     SEC JTAG FLASH(SJF) v 0.1      |
|     (S3C2440X & SMDK2440 B/D)      |
+------------------------------------+
Usage: SJF /f:<filename> /d=<delay>
> S3C2440X(ID=0x0032409d) is detected.

[SJF Main Menu]
 0:K9S1208 prog     1:28F128J3A prog   2:AM29LV800 Prog   3:SST39VF1601 Prog
 4:Memory Rd/Wr     5:Exit
Select the function to test:_
```

（5） 在出现的上图信息下，输入“3”，然后按“回车”（即 ENTER 键），选择烧写 SST39VF160，这个选项支持 SST39VF160 和 SST39VF1601。信息如下：

```
C:\WINDOWS\System32\cmd.exe

+------------------------------------+
|     SEC JTAG FLASH(SJF) v 0.1      |
|     (S3C2440X & SMDK2440 B/D)      |
+------------------------------------+
Usage: SJF /f:<filename> /d=<delay>
> S3C2440X(ID=0x0032409d) is detected.

[SJF Main Menu]
 0:K9S1208 prog     1:28F128J3A prog   2:AM29LV800 Prog   3:SST39VF1601 Prog
 4:Memory Rd/Wr     5:Exit
Select the function to test:3

[SST39VF160 Writing Program]
[Check SST39VF160]
Manufacture ID=  bf(0x00bf), Device ID(0x2782)=234b

Image Size:0h~b77ch

Available Target Offset:
    0x0,       0x40000,   0x80000,   0xc0000,
    0x100000,  0x140000,  0x180000,  0x1c0000,
Input target offset:_
```

接着在“Input target offset:”提示符下输入“0”，然后回车，这样，就开始进行 NOR FLASH 的烧写了。烧写要一段时间，烧写完成后，会自动退出。

```
C:\WINDOWS\System32\cmd.exe
Image Size:0h~9b00h

Available Target Offset:
    0x0,       0x40000,  0x80000,  0xc0000,
    0x100000, 0x140000, 0x180000, 0x1c0000,
Input target offset:0

SectorOffset=0x0
SectorSize   =0x1000
Erase the sector:0x0.
Sector Erase is started!
Start of the sector data writing.
0 100 200 300 400 500 600 700 800 900 a00 b00 c00 d00 e00 f00
End of the sector data writing!!!

SectorOffset=0x1000
SectorSize   =0x1000
Erase the sector:0x1000.
Sector Erase is started!
Start of the sector data writing.
0 100 200 300 400 500 600 700 800 900 a00 b00 c00 d00 e00 f00
End of the sector data writing!!!

SectorOffset=0x2000
SectorSize   =0x1000
Erase the sector:0x2000.
```

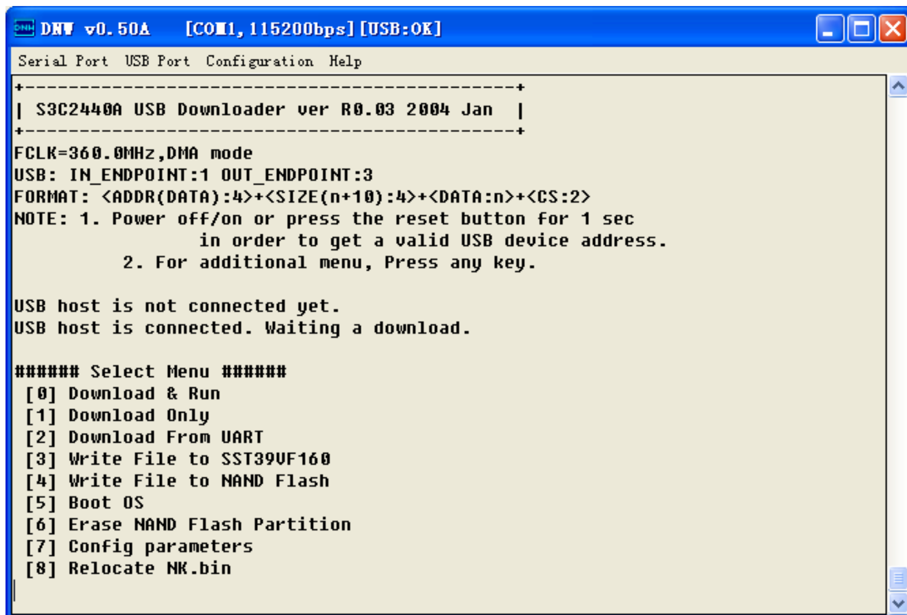
（6）烧写完成后，拔掉 JTAG 插头，设置好串口工具（DNW 或超级终端）的参数，上电复位就可以启动 BIOS 了。

第四章 烧写和启动 linux

这一章节的工作环境是 WINDOW。

4.1. 烧写 Linux 内核

（1）上电启动开发板，进入 BIOS 界面

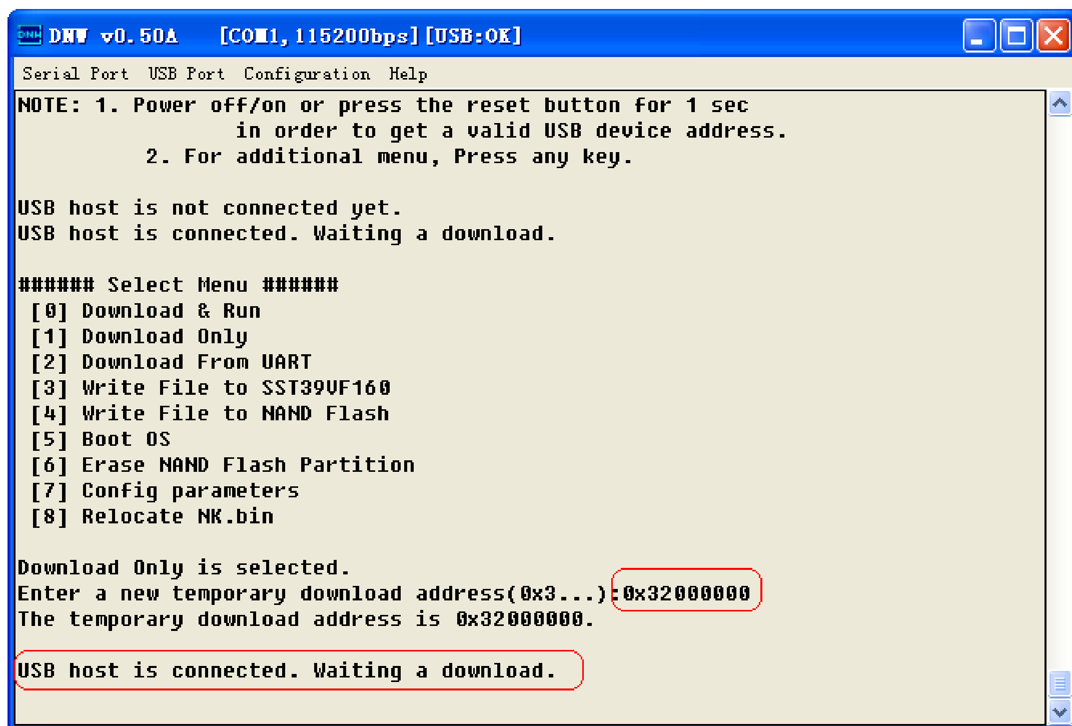


```
DNW v0.50A    [COM1,115200bps][USB:OK]
Serial Port  USB Port  Configuration  Help
+---------------------------------------------+
| S3C2440A USB Downloader ver R0.03 2004 Jan  |
+---------------------------------------------+
FCLK=360.0MHz,DMA mode
USB: IN_ENDPOINT:1 OUT_ENDPOINT:3
FORMAT: <ADDR(DATA):4>+<SIZE(n+10):4>+<DATA:n>+<CS:2>
NOTE: 1. Power off/on or press the reset button for 1 sec
                 in order to get a valid USB device address.
          2. For additional menu, Press any key.

USB host is not connected yet.
USB host is connected. Waiting a download.

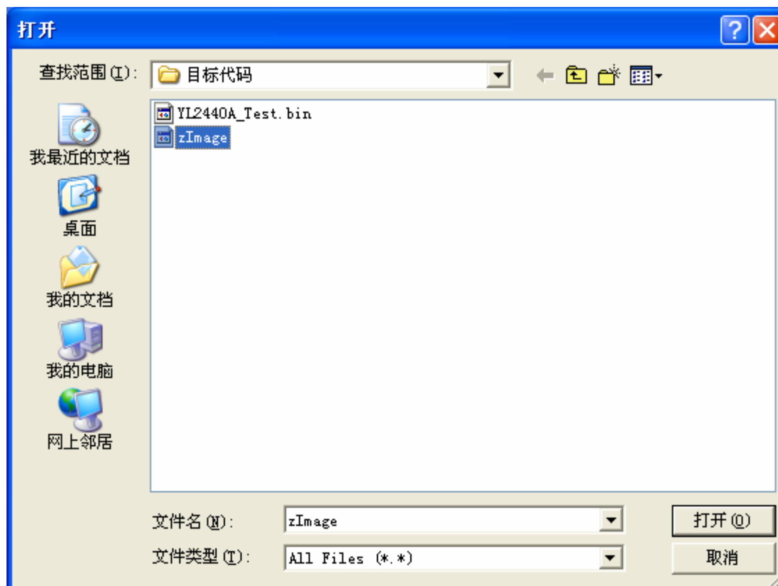
###### Select Menu ######
 [0] Download & Run
 [1] Download Only
 [2] Download From UART
 [3] Write File to SST39VF160
 [4] Write File to NAND Flash
 [5] Boot OS
 [6] Erase NAND Flash Partition
 [7] Config parameters
 [8] Relocate NK.bin
```

（2） USB DEVICE 的连接线，选择“1”，选择 USB 下载，接着出现“Enter a new temporary download address(0x3...):”的提示，在这个提示下输入程序下载的地址，这里**设置为 0x32000000**（不要与 BIOS 的地址 0x30100000 相冲突），同时要注意用 USB 下载要先在 PC 端装好驱动程序，保证 USB 连接好，有时 PC 端出现发现无法识别的 USB 设备时，可在 BIOS 中输入 ESC 取消下载，等几秒钟再输入“1”启动 USB 下载。



出现“USB host is connected”说明 USB 正确枚举了，这时可以通过 USB 下载了。

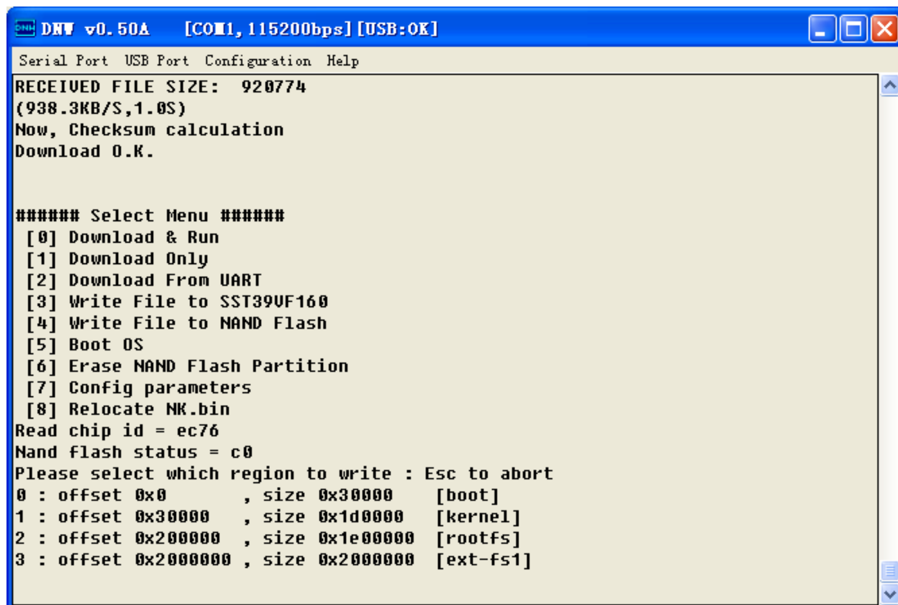
（3） 点击“USB Port”→”Transmit”选项，选择 zImage（这个映像文件在光盘目录的”目标代码“文件夹下）



点击“打开”，USB 下载开始。

下载结束后，会返回主功能菜单。

（4）下载成功后，在出现主功能菜单后，选择“4”，将出现如下界面。

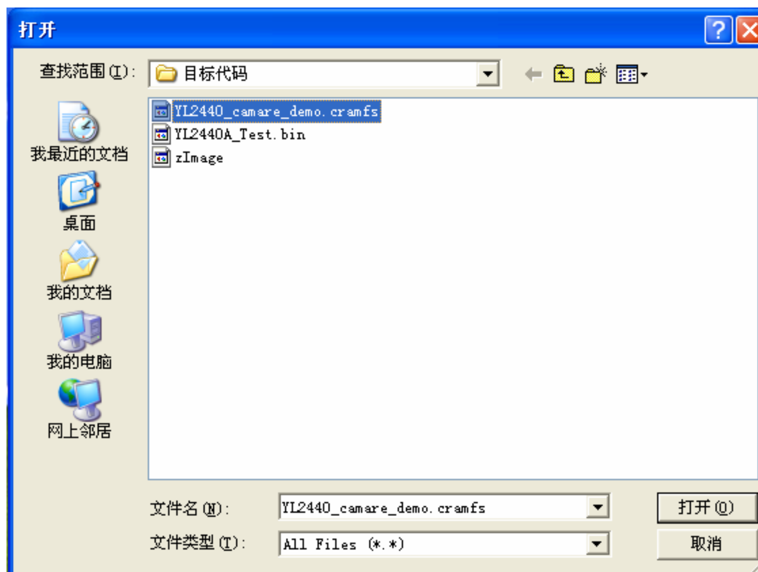


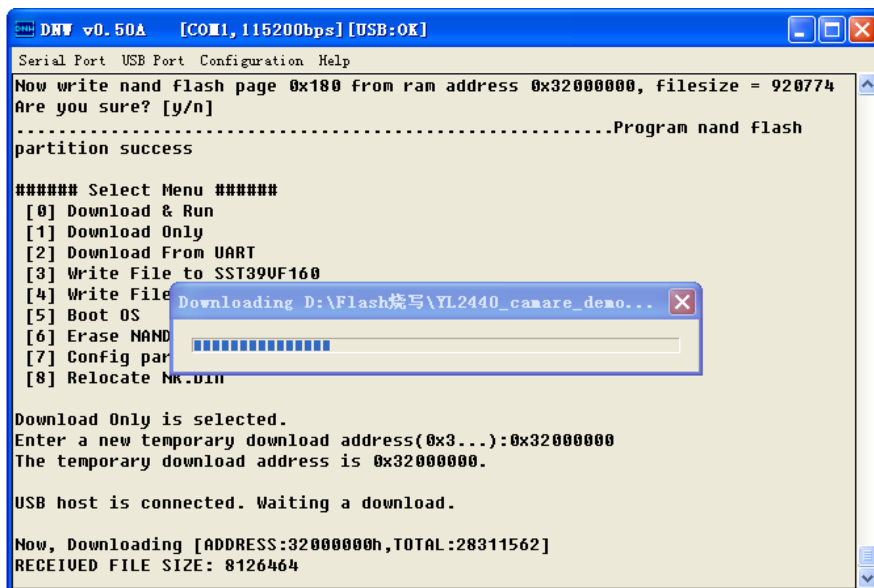
在出现上面的提示后，输入“1”，在接下来的提示输入“Y”，将 zImage 烧写到 NAND FLASH 的分区 1 中。烧写成功后，会自动进入主功能菜单。

4.2. 烧写根文件系统

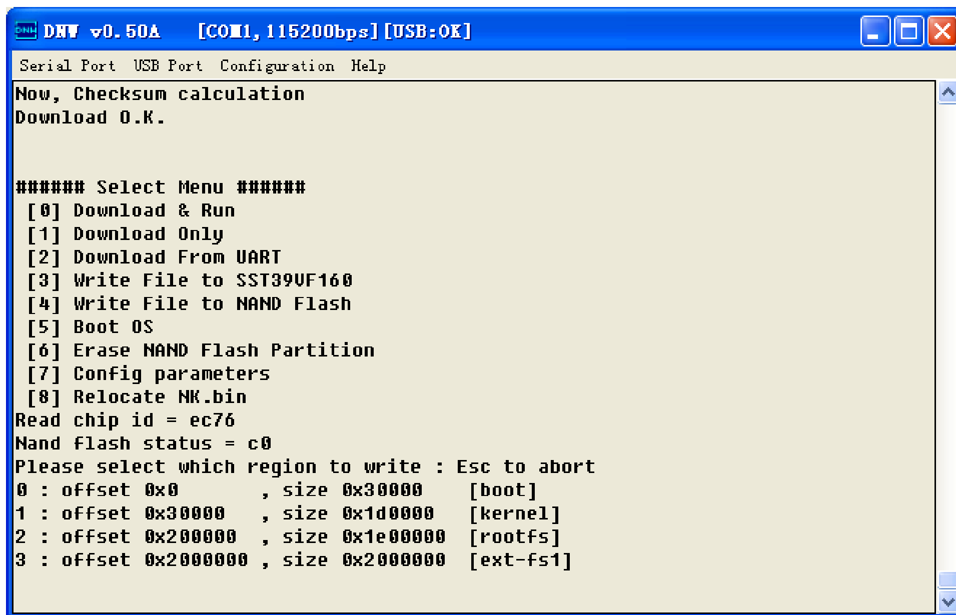
（1）烧写 Linux 根文件系统的前面步骤与上一章节《烧写 Linux 内核》的（1）和（2）步骤是一样的。

（2）当 USB DEVICE 成功枚举后，点击“USB Port”→”Transmit”选项，选择 YL2440_camare_demo.cramfs（这个映像文件在光盘目录的”目标代码“文件夹下）

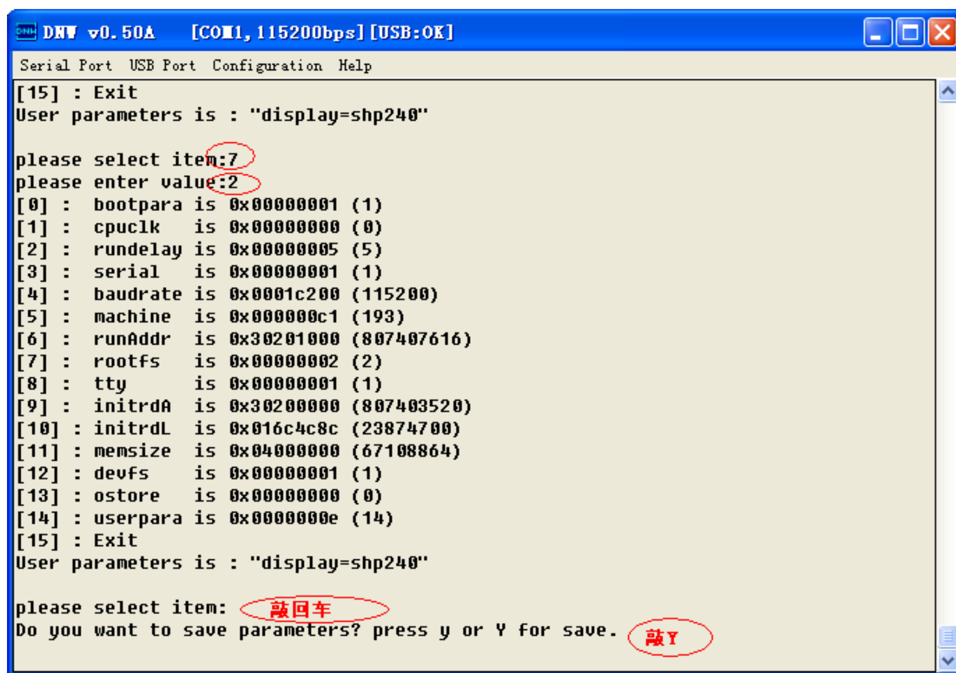




（4）下载成功后，在出现主功能菜单后，选择“4．Write file to Nand Flash”进行程序烧写，在接下来的分区选择中，选择分区**“2”**，出现提示输入后选择“Y”，将根文件系统烧写到 NAND FLASH 的分区 2 中，烧写成功后，会自动进入主功能菜单。



（5）设置根文件系统分区位置参数，这个参数主要是传递给内核的，在主功能菜单中，选择“7．Config paraments”进入系统参数配置，接着选择“7.rootfs”对 Linux 启动根目录进行配置，然后回车，在出现“please enter value:”提示下输入“2”设置 Linux 根文件系统为 2 分区，然后回车，然后再进行一次回车或者选择“15”退出，会提示“Do you want to save parameters? press y or Y for save.”，这时输入“y”，这样就将根文件系统所在的分区位置参数保存了。



（6）设置 LCD 显示参数，因为在板子的 LCD 标准配置是 Sharp 3.5”.

在 BIOS 的主功能菜单中，选择“8.Config paraments”，接着选择“14.Userpara”，然后回车，在“please enter value:”提示下输入“display=shp240”,，然后回车，再进行一次回车或者选择“15.Exit”退出，这时将提示“Do you want to save parameters? press y or Y for save.”，接着输入“Y”，这样就显示参数设置好了。

```
DNW v0.50A    [COM1,115200bps][USB:OK]
Serial Port  USB Port  Configuration  Help
[14] : userpara is 0x00000000 (0)
[15] : Exit

please select item:14
please enter value:display=shp240
[0] :  bootpara is 0x00000001 (1)
[1] :  cpuclk   is 0x00000000 (0)
[2] :  rundelay is 0x00000005 (5)
[3] :  serial   is 0x00000001 (1)
[4] :  baudrate is 0x0001c200 (115200)
[5] :  machine  is 0x000000c1 (193)
[6] :  runAddr  is 0x30201000 (807407616)
[7] :  rootfs   is 0x00000002 (2)
[8] :  tty      is 0x00000001 (1)
[9] :  initrdA  is 0x30200000 (807403520)
[10] : initrdL  is 0x016c4c8c (23874700)
[11] : memsize  is 0x04000000 (67108864)
[12] : devfs    is 0x00000001 (1)
[13] : ostore   is 0x00000000 (0)
[14] : userpara is 0x0000000e (14)
[15] : Exit
User parameters is : "display=shp240"

please select item:   敲回车
Do you want to save parameters? press y or Y for save.敲Y
```

4.3. 启动 Linux

将 Linux 内核和 Linux 根文件系统烧写好之后，接下来就是启动 Linux 了，启动 Linux 有两种方式，一种是通过 BIOS 的 5 号功能来启动，另一种是设置 Linux 自启动。

4.3.1. 通过 BIOS 的 5 号功能启动 Linux

开发板进入 BIOS 主功能菜单后，选择“5”从 Nand Flash 上启动程序，这时将启动 Linux,启动界面如下图。

```
DNW v0.50A    [COM4,115200bps][USB:OK]
Serial Port  USB Port  Configuration  Help
Nand flash status = c0
Set boot params = root=/dev/mtdblock3 init=/linuxrc
load_ramdisk=0 console=ttyS1,115200 mem=65536K devfs=mount
display=shp240
Load Kernel...
Linux version 2.4.18-rmk7-pxa1 (cgf@linuxserver.localdomain)
(gcc version 2.95.3 20010315 (release)) #2 六 7月 9 14:53:36
CST 2005

CPU: ARM/CIRRUS Arm920Tsid(wb) revision 0

Machine: Samsung-SMDK2440

On node 0 totalpages: 16384

zone(0): 16384 pages.

zone(1): 0 pages.

zone(2): 0 pages.

Kernel command line: root=/dev/mtdblock3 init=/linuxrc
load_ramdisk=0 console=ttyS1,115200 mem=65536K devfs=mount
display=shp240

set display type for YL-2440 platform.

Console: colour dummy device 80x30

Calibrating delay loop... 199.88 BogoMIPS

Memory: 64MB = 64MB total

Memory: 62408KB available (1534K code, 451K data, 72K init)

Dentry-cache hash table entries: 8192 (order: 4, 65536 bytes)
```

该 Linux 的根文件系统是带 QT 的界面，在 LCD 屏处，需要进行触摸屏校准才能进入 QT 界面，在屏上的十字叉中心点击，处理完后选择语言，点“下一步”到最后“Finish”就行了。在串口工具（DNW 或超级终端）处，按“Enter”键，将进入 Linux 的命令行模式下。如下图所示：

115200 - 超级终端

文件(F) 编辑(E) 查看(V) 呼叫(C) 传送(T) 帮助(H)

```
Please press Enter to activate this console. /bin/cp: will not create hard link
'/tmp/Applications/Network/modules' to directory '/tmp/Applications/AlarmServer'

/bin/cp: will not create hard link '/tmp/Documents' to directory '/tmp/Applicati
ons/qimpen'
Could not open mime.types file, "etc/mime.types", for reading
Could not read calibration: /etc/pointercal
Create pluginlibman in libqpe
Use QPEApplication's PluginLibraryManager
Setting up QCop to QPE/System
channel QPE/Card added
channel QPE/Server added
channel QPE/IME added
Create pluginlibman in libqpe
Use QPEApplication's PluginLibraryManager
QuickLauncher running
channel QPE/Application/quicklauncher added
channel QPE/QuickLauncher-63 added
Registered QPE/QuickLauncher-63 敲回车

ln: '/dev/touchscreen/0': File exists
[root@(none) /]# ls
Qtopia  bin  dev  etc  home  lib  linuxrc  proc  ramdisk  sbin  tmp  usr  var
[root@(none) /]#
```

已连接 0:01:52 ANSIW 115200 8-N-1 SCROLL CAPS NUM 捕 打印

4.3.2. Linux 的自启动

进入 BIOS 的主功能菜单后，选择“7”对启动参数进行配置，在接着出现的选项中输入“0”进行启动系统选择设置，然后回车，在出现“please enter value:”的提示下，输入 1 （1 为 Linux 系统，3 为 WIN CE 系统），然后回车，接着再进行一次回车（相当于输入 15 号退出功能），将会提示“Do you want to save parameters? press y or Y for save.”，这时输入“y”，这样就将该参数保存了。如下图所示

```
DNW v0.50A    [COM1,115200bps][USB:OK]
Serial Port  USB Port  Configuration  Help
[15] : Exit
User parameters is : "display=shp240"

please select item:0
please enter value:1
[0] :  bootpara is 0x00000001 (1)
[1] :  cpuclk   is 0x00000000 (0)
[2] :  rundelay is 0x00000005 (5)
[3] :  serial   is 0x00000001 (1)
[4] :  baudrate is 0x0001c200 (115200)
[5] :  machine  is 0x000000c1 (193)
[6] :  runAddr  is 0x30201000 (807407616)
[7] :  rootfs   is 0x00000002 (2)
[8] :  tty      is 0x00000001 (1)
[9] :  initrdA  is 0x30200000 (807403520)
[10] : initrdL  is 0x016c4c8c (23874700)
[11] : memsize  is 0x04000000 (67108864)
[12] : devfs    is 0x00000001 (1)
[13] : ostore   is 0x00000000 (0)
[14] : userpara is 0x0000000e (14)
[15] : Exit
User parameters is : "display=shp240"

please select item:
Do you want to save parameters? press y or Y for save.
```

选择“7”对启动参数进行配置，在接着出现的选项中输入“2”进行启动延时设置，然后回车，在出现“please enter value:”的提示下，输入 3（将延迟 3 秒后，自启动，如果输入的是 0，则没有设置为自启动），然后回车，接着再进行一次回车（相当于输入 15 号退出功能），将会提示“Do you want to save parameters? press y or Y for save.”，这时输入“y”，这样就将该参数保存了。如下图所示

```
DNW v0.50A    [COM1,115200bps][USB:OK]
Serial Port  USB Port  Configuration  Help
[15] : Exit
User parameters is : "display=shp240"

please select item:2
please enter value:3
[0] :  bootpara is 0x00000001 (1)
[1] :  cpuclk   is 0x00000000 (0)
[2] :  rundelay is 0x00000003 (3)
[3] :  serial   is 0x00000001 (1)
[4] :  baudrate is 0x0001c200 (115200)
[5] :  machine  is 0x000000c1 (193)
[6] :  runAddr  is 0x30201000 (807407616)
[7] :  rootfs   is 0x00000002 (2)
[8] :  tty      is 0x00000001 (1)
[9] :  initrdA  is 0x30200000 (807403520)
[10] : initrdL  is 0x016c4c8c (23874700)
[11] : memsize  is 0x04000000 (67108864)
[12] : devfs    is 0x00000001 (1)
[13] : ostore   is 0x00000000 (0)
[14] : userpara is 0x0000000e (14)
[15] : Exit
User parameters is : "display=shp240"

please select item:
Do you want to save parameters? press y or Y for save.
```

若还没设置根文件系统分区位置参数，一定要进行设置，这个参数主要是传递给内核的，在主功能菜单中，选择“7”进入系统参数配置，接着选择“7”对 Linux 启动根目录进行配置，然后回车，在出现“please enter value:”提示下输入“2”设置 Linux 根文件系统为 2 分区，然后回车，然后再进行一次回车或者选择“15”退出，会提示“Do you want to save parameters? press y or Y for save.”，这时输入“y”，这样就将根文件系统所在的分区位置参数保存了。

```
DNW v0.50A    [COM1,115200bps][USB:OK]
Serial Port  USB Port  Configuration  Help
[13] : ostore   is 0x00000000 (0)
[14] : userpara is 0x0000000e (14)
[15] : Exit
User parameters is : "display=shp240"

please select item:7
please enter value:2
[0] :  bootpara is 0x00000001 (1)
[1] :  cpuclk   is 0x00000000 (0)
[2] :  rundelay is 0x00000005 (5)
[3] :  serial   is 0x00000001 (1)
[4] :  baudrate is 0x0001c200 (115200)
[5] :  machine  is 0x000000c1 (193)
[6] :  runAddr  is 0x30201000 (807407616)
[7] :  rootfs   is 0x00000002 (2)
[8] :  tty      is 0x00000001 (1)
[9] :  initrdA  is 0x30200000 (807403520)
[10] : initrdL  is 0x016c4c8c (23874700)
[11] : memsize  is 0x04000000 (67108864)
[12] : devfs    is 0x00000001 (1)
[13] : ostore   is 0x00000000 (0)
[14] : userpara is 0x0000000e (14)
[15] : Exit
User parameters is : "display=shp240"

please select item:
```

设置 LCD 显示参数，在前面已介绍过一遍。因为在板子的 LCD 标准配置是 Sharp 3.5”. 在 BIOS 的主功能菜单中，选择“8”，接着选择“1 4 ”，然后回车，在“please enter value:” 提示下输入“display=shp240”，，然后回车，再进行一次回车或者选择“15”退出，这时将提示“Do you want to save parameters? press y or Y for save.”，接着输入“Y”，这样就显示参数设置好了。

```
DNW v0.50A    [COM1,115200bps][USB:OK]
Serial Port  USB Port  Configuration  Help
[14] : userpara is 0x00000000 (0)
[15] : Exit

please select item:14
please enter value:display=shp240
[0] :  bootpara is 0x00000001 (1)
[1] :  cpuclk   is 0x00000000 (0)
[2] :  rundelay is 0x00000005 (5)
[3] :  serial   is 0x00000001 (1)
[4] :  baudrate is 0x0001c200 (115200)
[5] :  machine  is 0x000000c1 (193)
[6] :  runAddr  is 0x30201000 (807407616)
[7] :  rootfs   is 0x00000002 (2)
[8] :  tty      is 0x00000001 (1)
[9] :  initrdA  is 0x30200000 (807403520)
[10] : initrdL  is 0x016c4c8c (23874700)
[11] : memsize  is 0x04000000 (67108864)
[12] : devfs    is 0x00000001 (1)
[13] : ostore   is 0x00000000 (0)
[14] : userpara is 0x0000000e (14)
[15] : Exit
User parameters is : "display=shp240"

please select item:   敲回车
Do you want to save parameters? press y or Y for save.敲Y
```

上电复位，在 3S 内，不进行任何操作，将自动启动 Linux,关于 Linux 的操作与上一节的 Linux 操作一样。

4.3. Linux 操作系统下的外围资源测试

在操作系统下主要测试网络，USB HOST 接口。

上电复位，不进行任何操作，将会自启动 Linux 系统（当然要保证 Linux 烧写到 NAND FLASH 的相应的分区中，关于 Linux 的烧写，请看前面的章节）。

YL2440 有两个网络接口，一个是 10M，一个是 100M

4.3.1. 10M 网络接口测试

用交叉网线将 PC 的网络接口与开发板的 10M 网络接口（CON3）连接起来。
Linux 启动后，按“回车”键，将进入命令行模式下。
输入命令：ifconfig eth0 192.168.3.100
输入命令：ifconfig eth0 可以看是否设置成功，如下图操作所示：

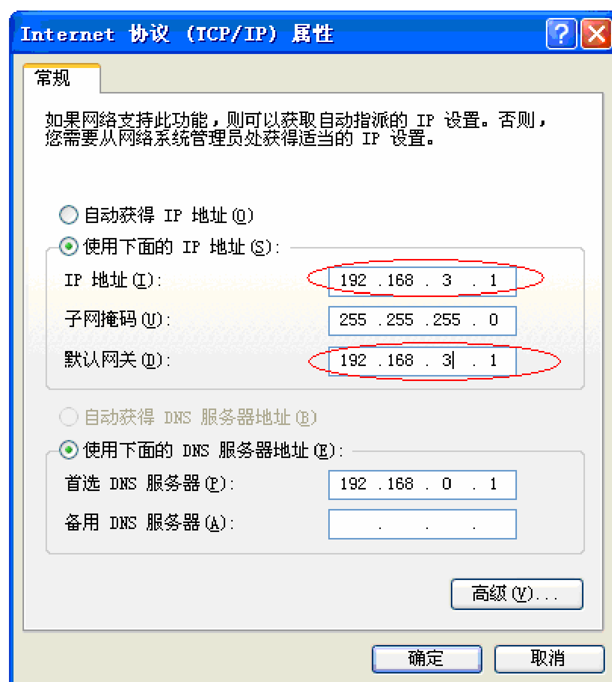
```
DNW v0.50A    [COM1,115200bps][USB:OK]
Serial Port  USB Port  Configuration  Help
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 iB)  TX bytes:0 (0.0 iB)

[root@(none) /]# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:00:C0:FF:EE:08
          inet addr:192.168.3.100  Bcast:192.168.3.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:0 (0.0 iB)  TX bytes:0 (0.0 iB)
          Interrupt:37 Base address:0x300

[root@(none) /]#
[root@(none) /]#
[root@(none) /]#
[root@(none) /]#
[root@(none) /]# ifconfig eth0 192.168.3.100
[root@(none) /]# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:00:C0:FF:EE:08
          inet addr:192.168.3.100  Bcast:192.168.3.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:4 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:857 (857.0 iB)  TX bytes:0 (0.0 iB)
          Interrupt:37 Base address:0x300

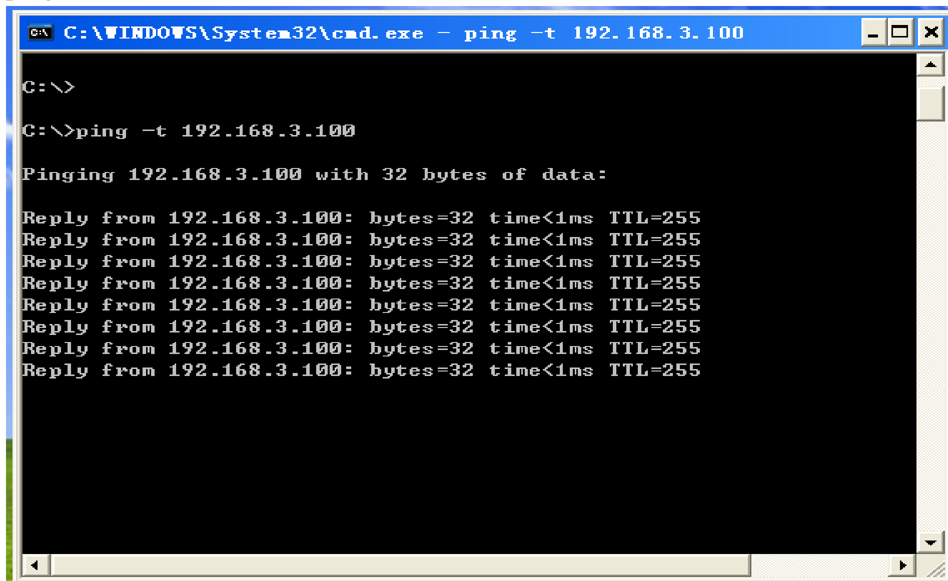
[root@(none) /]#
```

用 ifconfig 命令来配置开发板的 IP 地址，注意，开发板的 IP 地址要与 PC 的 IP 地址要设置为同一网段，这里设置为 192.168.3.100，同时 PC 上的网络放火墙要关闭。



IP 地址配置好后，打开 PC 端ＤＯＳ命令窗口，输入 PING 命令来 PING 开发板。

ping –t 192.168.3.100



出现上面的信息，就说明网络是正常的。

4.3.2. 100M 网络接口测试

用交叉网线将 PC 的网络接口与开发板的 100M NET 网络接口（JP2）连接起来。

配置时，用命令 ifconfig eth1 192.168.3.111，刚开始时插入 100M NET 网络接口时，PC 机上看是没有建立连接的，用命令配置完后才能看到连接，其他的过程跟 10M 网络一样了。

```
Serial Port  USB Port  Configuration  Help
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 iB)  TX bytes:0 (0.0 iB)

[root@(none) /]# ifconfig eth1 192.168.3.111
[root@(none) /]# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:00:C0:FF:EE:08
          inet addr:192.168.3.100  Bcast:192.168.3.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:0 (0.0 iB)  TX bytes:0 (0.0 iB)
          Interrupt:37 Base address:0x300

eth1      Link encap:Ethernet  HWaddr 00:12:34:56:80:49
          inet addr:192.168.3.111  Bcast:192.168.3.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:0 (0.0 iB)  TX bytes:0 (0.0 iB)
          Interrupt:42 Base address:0x300

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 iB)  TX bytes:0 (0.0 iB)

[root@(none) /]#
```

4.3.3. USB HOST 测试

将 U 盘插入 USB HOST 接口 CN3，Linux 将检测 U 盘，检测到后，自动给 U 盘分配一个设备节点，正确检测到 U 盘的信息如下：

115200 - 超级终端

文件(F) 编辑(E) 查看(V) 呼叫(C) 传送(T) 帮助(H)

```
channel QPE/IME added
Create pluginlibman in libqpe

ln: '/dev/touchscreen/0': File exists

[root@(none) /]#
[root@(none) /]# Use QPEApplication's PluginLibraryManager
QuickLauncher running
channel QPE/Application/quicklauncher added
channel QPE/QuickLauncher-63 added
Registered QPE/QuickLauncher-63

[root@(none) /]# hub.c: USB new device connect on bus1/1, assigned device number
 2
scsi0 : SCSI emulation for USB Mass Storage devices
  Vendor: USB-DISK  Model: FREEDIK-LWFORMAT  Rev: 3.01
  Type:   Direct-Access                     ANSI SCSI revision: 02
Attached scsi removable disk sda at scsi0, channel 0, id 0, lun 0
SCSI device sda: 260448 512-byte hdwr sectors (133 MB)
sda: Write Protect is off
Partition check:
 /dev/scsi/host0/bus0/target0/lun0: p1

[root@(none) /]#
```

已连接 0:01:26 ANSIW 115200 8-N-1 SCROLL CAPS NUM 捕 打印

正确检测到 U 盘后，接下来就可以挂接 U 盘了。

挂接 U 盘的命令如下：

mkdir /tmp/1

mount -t vfat /dev/scsi/host0/bus0/target0/lun0/part1   /tmp/1

```
115200 - 超级终端
文件(F) 编辑(E) 查看(V) 呼叫(C) 传送(T) 帮助(H)

the contents of that file.  If you want more assurance that the contents are
truly unrecoverable, consider using shred.

Report bugs to <bug-coreutils@gnu.org>.
[root@(none) /]# rm -v /tmp/1
rm: cannot remove '/tmp/1': Is a directory
[root@(none) /]# rm -drv /tmp/1
removed directory: '/tmp/1'
[root@(none) /]# ls
Qtopia  bin  dev  etc  home  lib  linuxrc  proc  ramdisk  sbin  tmp  usr  var
[root@(none) /]# cd tmp
[root@(none) tmp]# ls
Applications  Settings  qcop-msg-qpe             qtembedded-unknown
Documents     disk3     qcop-msg-quicklauncher   udisk
[root@(none) tmp]# cd ..
[root@(none) /]# ls
Qtopia  bin  dev  etc  home  lib  linuxrc  proc  ramdisk  sbin  tmp  usr  var
[root@(none) /]# mkdir /tmp/1/
[root@(none) /]# mount -t vfat /dev/scsi/host0/bus0/target0/lun0/part1 /tmp/1
modprobe: modprobe: Can't open dependencies file /lib/modules/2.4.18-rmk7-pxa1/
odules.dep (No such file or directory)
modprobe: modprobe: Can't open dependencies file /lib/modules/2.4.18-rmk7-pxa1/
odules.dep (No such file or directory)
[root@(none) /]#

已连接 0:16:01 ANSIW  115200 8-N-1  SCROLL  CAPS  NUM  捕  打印
```

这样就将 U 盘挂接到/tmp/1 目录下，这时我们就可以进入/tmp/1 目录，对 U 盘进行读写，删除，建立文件等操作了。

```
文件(F) 编辑(E) 查看(V) 呼叫(C) 传送(T) 帮助(H)
Applications  Settings  qcop-msg-qpe          qtembedded-unknown
[root@(none) scsi]# ls0xea000045 [Correct ON

[root@(none) /]# ls
Qtopia  bin  dev  etc  home  lib  linuxrc  proc  ramdisk  sbin  tmp  usr  var
[root@(none) /]# mkdir /tmp/1/
[root@(none) /]# mount -t vfat /dev/scsi/host0/bus0/target0/lun0/part1 /tmp/1
modprobe: modprobe: Can't open dependencies file /lib/modules/2.4.18-rmk7-pxa1/m
odules.dep (No such file or directory)
modprobe: modprobe: Can't open dependencies file /lib/modules/2.4.18-rmk7-pxa1/m
odules.dep (No such file or directory)
[root@(none) /]# cd /tmp/1
[root@(none) 1]# ls
??.doc                              RIT300??????
?????                               YL44B0X????.rar
????? (2)                           YL44B0X_Test
??????????.doc                      YL44B0X_V10end.pcb
ARMorder.txt                        divs
EDB9307_schematics_v_2.pdf          fs44b0lib
FS44B0XP??????V2.0.2.pdf            fs44b0xv_v10
MT48LCm32B2.pdf                     lcddrive.c
Pcai.rar                            m44b0lib
R8610START                          order.txt
R8610START.rar                      r8610.sch
R8610_IDEserver_CUT_power.sch       vw2010_hdwr_desc_0.95.pdf
R8610lib                            vw_card.rar
[root@(none) 1]#
已连接 0:16:4( ANSIW  115200 8-N-1  SCROLL  CAPS  NUM  捕  打印
```

4.3.4. IDE 硬盘挂接

先要将 IDE 硬盘的数据线与 YL2440 的 IDE 接口 J7 连接好，打开 IDE 硬盘电源，打开开发板电源，然后启动 Linux，Linux 将检测到 IDE 硬盘的信息，显示信息如下：

DNW v0.50A [COM4,115200bps][USB:x]

Serial Port USB Port Configuration Help

```
RAMDISK driver initialized: 16 RAM disks of 4096K size 1024
blocksize

Uniform Multi-Platform E-IDE driver Revision: 6.31

ide: Assuming 50MHz system bus speed for PIO modes; override
with idebus=xx

hda: ST32132A, ATA DISK drive

ide0: unexpected interrupt, status=0x51, count=1

ide0 at 0xc4880020-0xc4880027,0xc488001c on irq 1

hda: 4127760 sectors (2113 MB) w/120KiB Cache, CHS=4095/16/63

Partition check:

 /dev/ide/host0/bus0/target0/lun0: [PTBL] [1023/64/63] p1

eth0: cs8900 rev K(3.3 Volts) found at 0xd0000300

cs89x0 media RJ-45, IRQ 37
```

出现上面的信息后，说明 IDE 硬盘的分区被正确检测到了。

现在就可以挂接硬盘分区了。

请注意,要看一下在 Linux 下挂接的设备具体路径,你可以进入/dev 目录下找 IDE 子目录,再看下面的设备,在 IDE 挂接成功时,用 ls /dev/ide 命令可以看到一个文件夹 host0 如下图:

115200 - 超级终端

文件(F) 编辑(E) 查看(V) 呼叫(C) 传送(T) 帮助(H)

```
[root@(none) ide]# ls
host0
[root@(none) ide]# cd ..
[root@(none) dev]# ls
cdroms   fb           input  mtd       pts     scsi         ttsA   ura
console  fb0          kmem   mtdblock  pty     shm          tty    usb
cua      full         loop   nbd       random  sound        tty0   vc
cuaA     h3600_tsraw  mem    null      rd      touchscreen  tty1   vcc
discs    i2c          misc   port      root    ts           tty2   vid
dsp      ide          mixer  ptmx      rtc     tts          ttyS0  zer
[root@(none) dev]# cd ..
[root@(none) /]# ls
Qtopia  bin  dev  etc  home  lib  linuxrc  proc  ramdisk  sbin  tmp  u
[root@(none) /]# cd /dev/ide
[root@(none) ide]# ls
host0
[root@(none) ide]# ls /dev/ide
host0
[root@(none) ide]#
```

已连接 0:33:40 ANSIW 115200 8-N-1 SCROLL CAPS NUM 捕 打印

mkdir /tmp/2

mount -t vfat /dev/ide/host0/bus0/target0/lun0/part1 /tmp/2

这样就将 IDE 硬盘分区挂接到/tmp/2 目录下，这时我们就可以进入/tmp/2 目录，对硬盘进行操作了。

```
115200 - 超级终端
文件(F) 编辑(E) 查看(V) 呼叫(C) 传送(T) 帮助(H)
t@(none) /]#
t@(none) /]#
t@(none) /]#
t@(none) /]# ls
ia  bin  dev  etc  home  lib  linuxrc  proc  ramdisk  sbin  tmp  usr  var
t@(none) /]# mkdir /tmp/2
t@(none) /]#
t@(none) /]#

t@(none) /]# mount -t vfat /dev/ide/host0/bus0/target0/lun0/part1 /tmp/2
robe: modprobe: Can't open dependencies file /lib/modules/2.4.18-rmk7-pxa1/m
es.dep (No such file or directory)
robe: modprobe: Can't open dependencies file /lib/modules/2.4.18-rmk7-pxa1/m
es.dep (No such file or directory)
t@(none) /]# cd /tmp/2
t@(none) 2]# ls
te            bootlog.prv  ghost       pdos.def      suhdlog.dat
ocuments      bootlog.txt  ide         pqtmp.fil     system.1st
ram Files     command.com  io.sys      recycled      videorom.bin
torage        config.sys   msdos.---   scandisk.log  windows
log           detlog.txt   msdos.sys   setup
exec.bat      frunlog.txt  netlog.txt  setuplog.txt       硬盘上的文件信息
t@(none) 2]#
已连接 0:42:06 ANSIW  115200 8-N-1  SCROLL  CAPS  NUM  捕  打印
```

4.3.5. 串口 3 测试

首先进入 Linux 命令行模式，输入命令：stty –F /dev/ttsA/0 115200 (此命令是对外扩 16550 串口的设置)

接着输入命令：

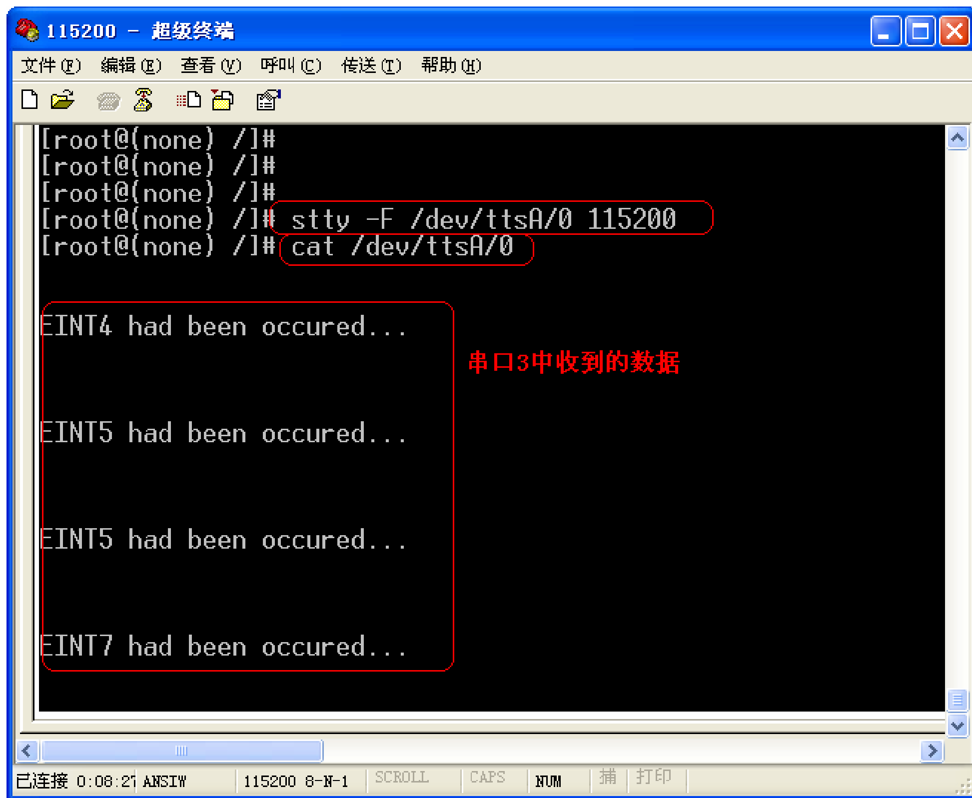
cat /dev/ttsA/0

输入上面两条命令后，用交叉串口线，将 PC 的串口与 YL2440 的串口 3（P3）相连，同时打开另一个串口工具，将 PC 端的串口设置好。

串口工具的参数：波特率 115200，8 位，无奇偶位，停止位 1，无硬件流。

设置好后，就可以该串口工具里输入字符或数字，然后回车，这时，就可以在 Linux 的那个串口工具（超级终端）里看到刚刚输入的字符或数字。

在串口 3 上发出的数据就可以在 Linux 中看到了(超级终端中)，如下图所示：



4.3.6. Camera 测试

进入 tmp 目录下,将光盘中的目标代码文件下的 testcamera 文件传到板子上运行即可启动摄像头,具体操作如下:

在 shell 下执行,

cd /tmp

rz

利用 zmodem 协议将 testcamera 发送到板子

ls

chmod 755 testcamera

./testcamera –u s3c2440a

就可以启动摄像头测试程序，启动成功后会显示如下信息，

```
115200 - 超级终端
文件(F) 编辑(E) 查看(V) 呼叫(C) 传送(T) 帮助(H)
testcamera
[root@(none) tmp]# ./testcamera -o 1
[root@(none) tmp]# cdmand [-pVv] command [

[root@(none) tmp]# ls
Applications  Settings  pointercal   qcop-msg-quicklauncher  testcamera
Documents     disk3     qcop-msg-qpe  qtembedded-unknown      udisk
[root@(none) tmp]# chmod 755 testcamera
[root@(none) tmp]# ./testcamera -u s3c2440a
240x320, 16bpp
CAMERA : UPLL 96000000  UCLK 48000000 CAMCLK 24000000
camera power level 0, reset level 1
Open camera success
open i2c device...
set slave address to 0x30 success!
manufactory ID is 0x7fa2
product ID is 0x9652
max width 800, height 600
min width 160, height 120
current width 640, height 480
capture width 640, height 480
current palette 13
allocate memory for camera buffer.

buffer at 0x4019b008
now start capture...
press Esc key to exit, 'c' to save picture, 'd' to enable/disable display
_
已连接 0:03:35 TTY 115200 8-N-1 SCROLL CAPS NUM 捕 打印
```

这时在 LCD 上会动态显示摄像头捕捉到的画面，在控制台下按 d 键可以使能/禁止显示，c 键可将捕捉的画面保存到 jpg 文件里，Esc 键可退出测试。

coma - 超级终端

文件(F) 编辑(E) 查看(V) 呼叫(C) 传送(T) 帮助(H)

```
max width 1280, height 1024
min width 88, height 72Invalid ioctl parameters.

current width 640, height 480
VIDIOCSCAPTURE fail
VIDIOCSPICT fail
current palette 13
current input format 15, output format 16
current camera buffer size 960000, total frames 1
mmap camera ok.

buffer at 0x4020c000
now start capture...
press Esc key to exit, 'c' to save picture, 'd' to enable/disable display
disable display
enable display
save picture...
PXA_CAMERA: pxa_camera_close
[root@(none) tmp]# ls
Applications            images                    qtembedded-unknown
Documents               jpgfile0000.jpg           resolv.conf
Settings                qcop-msg-qpe              sdcard
flashdisk               qcop-msg-quicklauncher    udisk
[root@(none) tmp]# _
```

已连接 0:40:26 ANSIW 115200 8-N-1 SCROLL CAPS NUM 捕 打印

上图显示的 jpgfile0000.jpg 就是测试程序保存的图片文件。

第五章 运行 WINCE

本章节，工作在 WINDOW2000/XP 环境下。

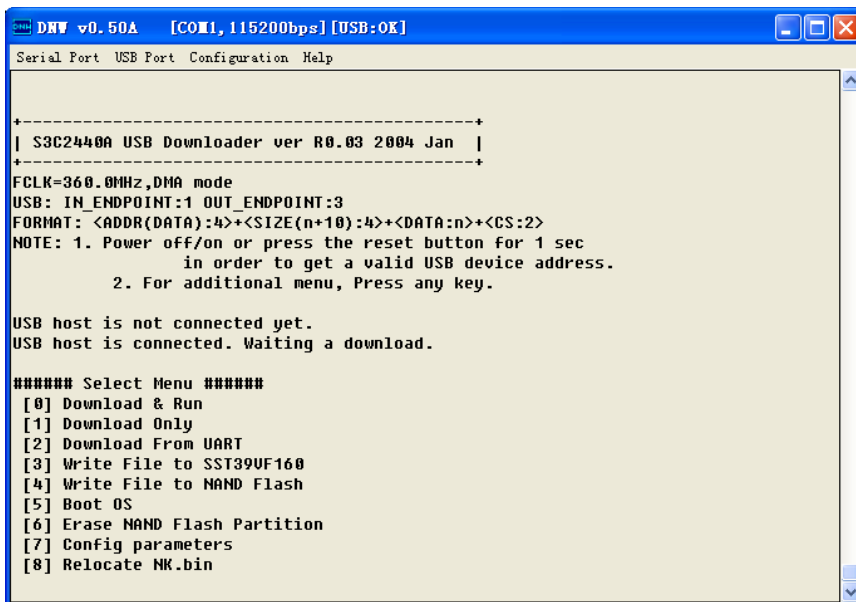
5.1. 下载运行 WINCE

这小节主要介绍通过 USB 下载 NK.nb0 到 SDRAM 中来运行，下载和运行地址为 0x30200000。

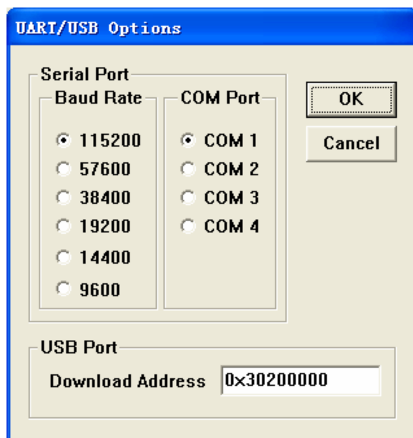
步骤如下：

（1） 用串口线将 PC 机和开发板的串口 2（P2，中间那个串口）连接起来，打开 DNW，设置好串口参数：波特率 115200，8 位，无奇偶位，停止位 1，无硬件流。
接着连接好 USB DEVICE，打开电源。
如果选配的 LCD 套件的话，连接好它。

（2） 上电后将启动 BIOS，按任意键，不要让 BIOS 自启动 Linux，进入 BIOS 的主功能菜单模式下：



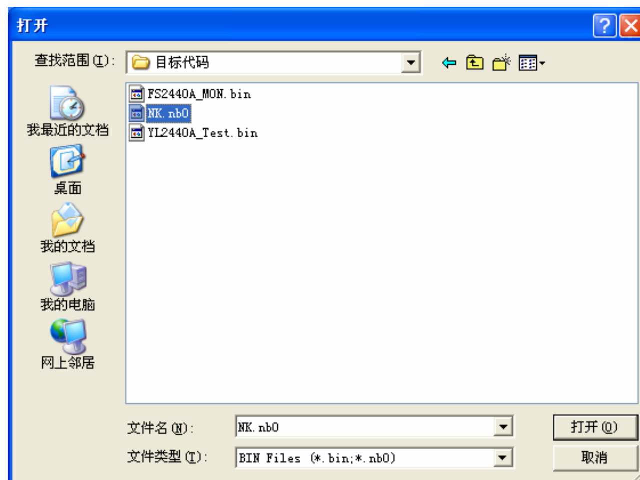
（3） 先设置好 USB 下载地址，打开 DNW 的“Configuration” -> “options”，将 USB 的下载地址设置为 0x30200000，设置如下：



接着点击“OK”

（4） 设置好 USB 下载地址后，在主功能菜单中，选择“0”，将进行 USB 下载，如果出现“USB host is connected. Waiting a download.”提示符号，说明 USB 正确检测到了（如果没有出现，请按“ESC”取消，重新进入主菜单，然后在选择“0”，一直到 USB 正确检测到了为止，如果是第一次使用，还需要安装 USB 驱动）。

（5） 点击 DNW 的“USB Port” → “T ransmit”，然后选择要传输的 NK.nb0（这个文件在光盘目录的“目标代码”文件夹中），然后点击“打开”按钮。



（6）下载结束，会自动运行刚刚下载的 NK.nb0，

WINCE 运行后，会在串口和 LCD 界面处有信息显示。
串口处的打印信息如下：

```
Windows CE Kernel for ARM (Thumb Enabled) Built on Mar 13 2003 at 22:52:56
ProcessorType=0920    Revision=0
sp_abt=ffff5000 sp_irq=ffff2800 sp_undef=ffffc800 OEMAddressTable = 8c2013bc

Windows CE Firmware Init
INFO: Initializing system interrupts...
INFO: Initializing system clock(s)...
INFO: Initializing driver globals area...
SDMMC config set rGPGCON: 86a9aa
OEMInit Done...
Sp=ffffc7cc
```

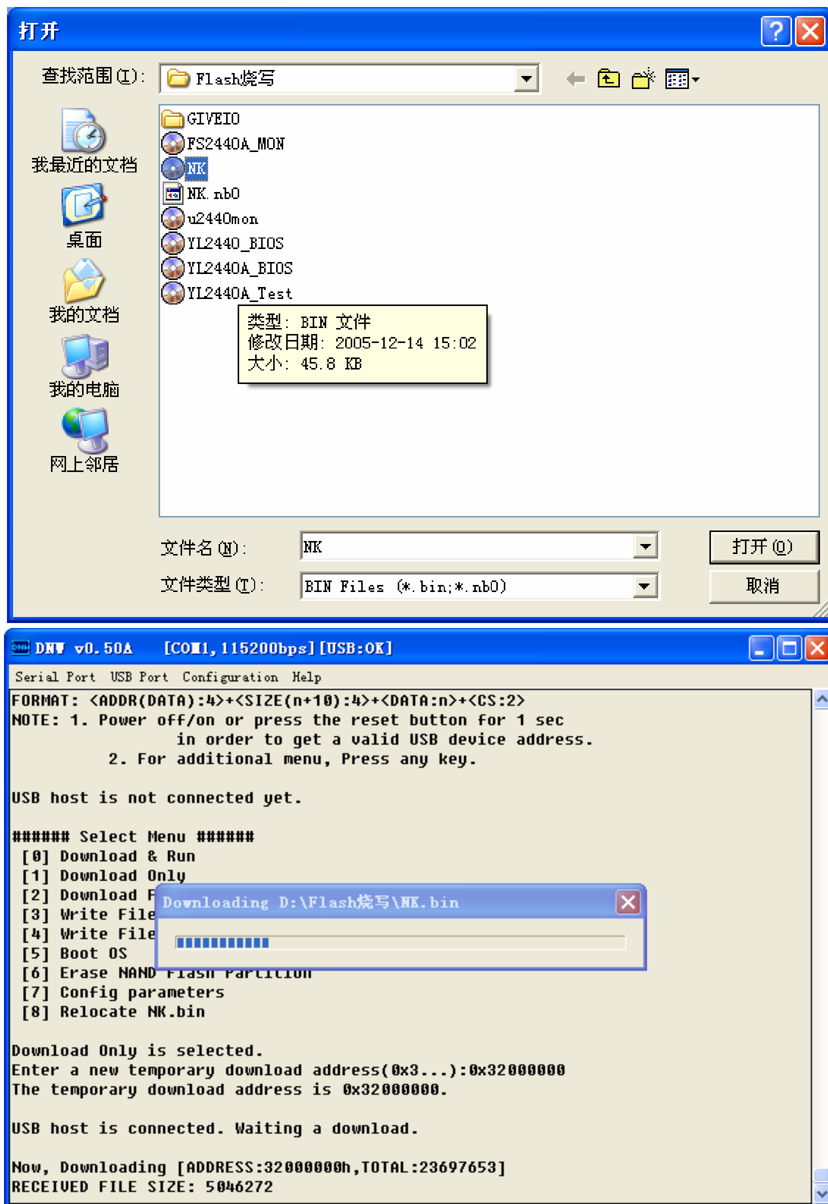
上面只列出了部分信息。

5.2. WINCE 的烧写

这一章主要介绍，如何烧写 WINCE 的 NK.bin,然后设置自启动 WINCE。

（1）进入 BIOS 的主功能菜单，选择“1”，选择 USB 下载，接着出现“Enter a new temporary
download address(0x3...):”的提示，在这个提示下输入程序下载的地址，这里设置为 0x32000000，同时要注意用 USB 下载要先在 PC 端装好驱动程序，保证 USB 连接好，有时 PC 端出现发现无法识别的 USB 设备时，可在 BIOS 中输入 ESC 取消下载，等几秒钟再输入“1”启动 USB 下载。

（2） 点击“USB Port”→”Transmit”选项，选择 NK.bin（这个映像文件在光盘目录的”目标代码“文件夹下）



（3）下载结束后，会自动返回到 BIOS 的主功能菜单，这时选择“8”对 bin 文件进行解压，如下图：

```
DNW v0.50A    [COM1,115200bps][USB:OK]
Serial Port  USB Port  Configuration  Help
RECEIVED FILE SIZE:23697653
(886.3KB/S,26.7S)
Now, Checksum calculation
Download O.K.


###### Select Menu ######
 [0] Download & Run
 [1] Download Only
 [2] Download From UART
 [3] Write File to SST39VF160
 [4] Write File to NAND Flash
 [5] Boot OS
 [6] Erase NAND Flash Partition
 [7] Config parameters
 [8] Relocate NK.bin
Found single BIN signature.      输入"8"
Image start is 0x8c200000, length is 0x016c4c8c
REC start 0x8c200000, length 0x4
REC start 0x8c200040, length 0x8
REC start 0x8c201000, length 0x63d44
REC start 0x8c264d44, length 0x2b8
REC start 0x8c265000, length 0x2000
REC start 0x8c267000, length 0x77474
REC start 0x8c2de474, length 0x7b14
REC start 0x8c2e6000, length 0x2fffc
```

操作完成后出现以下界面，选择“N”。

```
DNW v0.50A    [COM1,115200bps][USB:OK]
Serial Port  USB Port  Configuration  Help
REC start 0x8cc86000, length 0x5ffc
REC start 0x8cc8c000, length 0x2ffc
REC start 0x8cc8f000, length 0xbffc
REC start 0x8cc9b000, length 0x3ffc
REC start 0x8cc9f000, length 0x10ffc
REC start 0x8ccb0000, length 0x334
REC start 0x8ccb1000, length 0x3ffc
REC start 0x8ccb5000, length 0x2f08
REC start 0x8ccb8000, length 0xbb4
REC start 0x8ccb9000, length 0x28ebc
REC start 0x8cce1ebc, length 0xa9980
REC start 0x8cd8b83c, length 0x3b798
REC start 0x8cdc6fd4, length 0x1f74c
REC start 0x8cde6720, length 0xa03930
REC start 0x8d7ea050, length 0x373bc
REC start 0x8d82140c, length 0x2b000
REC start 0x8d84c40c, length 0x3c660
REC start 0x8d888a6c, length 0x3a560
REC start 0x8d8c2fcc, length 0x54
    ROMHDR 0x8d8c2fcc, 0x8d8c2fcc
    dllfirst  0x01e801eb, dlllast  0x02000000
    physfirst 0x8c200000, physlast 0x8d8c4c8c
REC start 0x8d8c3020, length 0x1c6c
real entry is 0x8c201000, physical address is 0x30201000
do you want to run it? [y/n]
```

再选择“4.Write File to NAND Flash”，输入“3”，出现如下界面：

```
DNW v0.50A    [COM1,115200bps][USB:OK]
Serial Port  USB Port  Configuration  Help
Read chip id = ec76
Nand flash status = c0
.Save boot params success.

###### Select Menu ######
 [0] Download & Run
 [1] Download Only
 [2] Download From UART
 [3] Write File to SST39VF160
 [4] Write File to NAND Flash
 [5] Boot OS
 [6] Erase NAND Flash Partition
 [7] Config parameters
 [8] Relocate NK.bin
Read chip id = ec76
Nand flash status = c0
Please select which region to write : Esc to abort
0 : offset 0x0       , size 0x30000    [boot]
1 : offset 0x30000   , size 0x1d0000   [kernel]
2 : offset 0x200000  , size 0x1e00000  [rootfs]
3 : offset 0x2000000 , size 0x2000000  [ext-fs1]
Now write nand flash page 0x10000 from ram address 0x30200000, filesize = 23874700
Are you sure? [y/n]
.................................................................................
.................................................................................
..............................
```

再接下来的提示输入”Y“，将 NK.bin 烧写到 NAND FLASH 的分区 3 中，烧写成功后，会自动进入主功能菜单。

5.3. WINCE 的自启动

进入 BIOS 的主功能菜单后，选择“7”，在接着出现的选项中输入“0”，然后回车，在出现“please enter value:”的提示下，输入 3(1：启动 linux,3：启动 wince)，然后回车，接着再回车，将会提示“Do you want to save parameters? press y or Y for save.”，这时输入“y”，这样就将该参数保存了。

```
DNW v0.50A    [COM1,115200bps][USB:OK]
Serial Port  USB Port  Configuration  Help
[13] : ostore   is 0x00000000 (0)
[14] : userpara is 0x0000000e (14)
[15] : Exit
User parameters is : "display=shp240"

please select item:0
please enter value:3
[0] :  bootpara is 0x00000003 (3)
[1] :  cpuclk   is 0x00000000 (0)
[2] :  rundelay is 0x00000005 (5)
[3] :  serial   is 0x00000001 (1)
[4] :  baudrate is 0x0001c200 (115200)
[5] :  machine  is 0x000000c1 (193)
[6] :  runAddr  is 0x30201000 (807407616)
[7] :  rootfs   is 0x00000002 (2)
[8] :  tty      is 0x00000001 (1)
[9] :  initrdA  is 0x30200000 (807403520)
[10] : initrdL  is 0x016c4c8c (23874700)
[11] : memsize  is 0x04000000 (67108864)
[12] : devfs    is 0x00000001 (1)
[13] : ostore   is 0x00000000 (0)
[14] : userpara is 0x0000000e (14)
[15] : Exit
User parameters is : "display=shp240"

please select item:
```

其它各参数要是在前面的章节中已绍设置，若有不同，请参照上图进行调整设置。上电复位，在 3S 内，不进行任何操作，将自动启动 WINCE。

第六章 LINUX 内核编译

6.1.安装编译工具

编译 LINUX 前,要先安装交叉编译工具 TOOLCHAIN,我们随板光盘带了两个编译工具。

一个是 cross-2.95.3.tar.bz2，安装此编译器只需要在/usr/local 目录下建一个 arm 的目录，进入 arm 目录再执行解包命令（tar zxvf /mnt/cdrom/linux/toolchain/cross-2.95.3.tar.bz2）即可，之后可编辑/etc/bashrc 文件，在最后增加路径 export PATH=/usr/local/arm/2.95.3/bin:$PATH，以后编译内核或其他应用程序均可用 arm-linux- 来指定交叉编译器。

另一个编译器是 MIZI 提供的 ARMV4l 安装包，它的安装分几步，需要用 rpm –ivh xxxx 的方式来安装，必须的包有 binutils、gcc、glibc，有其他应用时还会需要安装另外的工具，如内核头文件、c++、libjpeg、zlib 等，它们都会安装在/opt/host/armv4l 目录下，不需要再手工设置路径。

6.2.解压 LINUX 内核

将我们提供的 LINUX 内核压缩包解包到您所想要的目录下,如:

tar jxvf /mnt/cdrom/linux/s3c2440_kernel2.4.18.tar.bz2

当然,你可以将我们提供的光盘中的“Linux 内核源码和工具/内核源码”包中的 s3c2440_kernel2.4.18.tar.bz2 文件拷贝到自定的目录中，直截解压在当前目录下，

tar jxvf s3c2440_kernel2.4.18.tar.bz2

6.3.编辑 MAKEFILE 文件

进入解压后的目录，如生成的目录是“s3c2440_kernel2.4.18_rel”，
如输入:cd s3c2440_kernel2.4.18_rel
先编辑 Makefile 文件

```
2.4.18-rmk7/fs/file.c
2.4.18-rmk7/fs/ramfs/
2.4.18-rmk7/fs/ramfs/Makefile
2.4.18-rmk7/fs/ramfs/inode.c
2.4.18-rmk7/fs/dnotify.c
2.4.18-rmk7/fs/exec.c
2.4.18-rmk7/fs/coda/
2.4.18-rmk7/fs/coda/coda_linux.c
2.4.18-rmk7/fs/coda/cache.c
2.4.18-rmk7/fs/coda/Makefile
2.4.18-rmk7/fs/coda/pioctl.c
2.4.18-rmk7/fs/coda/cnode.c
2.4.18-rmk7/fs/coda/inode.c
2.4.18-rmk7/fs/coda/upcall.c
2.4.18-rmk7/fs/coda/dir.c
2.4.18-rmk7/fs/coda/symlink.c
2.4.18-rmk7/fs/coda/psdev.c
2.4.18-rmk7/fs/coda/sysctl.c
2.4.18-rmk7/fs/coda/file.c
2.4.18-rmk7/init/
2.4.18-rmk7/init/version.c
2.4.18-rmk7/init/main.c
2.4.18-rmk7/README
2.4.18-rmk7/rm
2.4.18-rmk7/user/
2.4.18-rmk7/user/helloworld/
2.4.18-rmk7/user/helloworld/Makefile
2.4.18-rmk7/user/helloworld/env.sh
2.4.18-rmk7/user/helloworld/hello.c
2.4.18-rmk7/user/helloworld/qhello.cpp
2.4.18-rmk7/user/helloworld/hello
2.4.18-rmk7/user/helloworld/test.c
2.4.18-rmk7/kernel_2410.cfg
2.4.18-rmk7/zImage
2.4.18-rmk7/kernel_2410.cfg.old
[root@localhost work]# cd 2.4.18-rmk7/
[root@localhost 2.4.18-rmk7]# vi Makefile
```

设置好编译器再保存退出

输入命令：vi Makefile

找到 CROSS_COMPILE = opt/host/armv41/bin/armv41-unknown-linux- 这行，将它改为

CROSS_COMPILE = arm-linux-

```
VERSION = 2
PATCHLEVEL = 4
SUBLEVEL = 18
EXTRAVERSION = -rmk7-pxa1

KERNELRELEASE=$(VERSION).$(PATCHLEVEL).$(SUBLEVEL)$(EXTRAVERSION)

#ARCH := $(shell uname -m | sed -e s/i.86/i386/ -e s/sun4u/sparc64/ -e s/arm.*/arm/ -e s/sa110/arm/)
ARCH := arm
KERNELPATH=kernel-$(shell echo $(KERNELRELEASE) | sed -e "s/-//g")

CONFIG_SHELL := $(shell if [ -x "$$BASH" ]; then echo $$BASH; \
          else if [ -x /bin/bash ]; then echo /bin/bash; \
          else echo sh; fi ; fi)
TOPDIR   := $(shell /bin/pwd)

HPATH           = $(TOPDIR)/include
FINDHPATH       = $(HPATH)/asm $(HPATH)/linux $(HPATH)/scsi $(HPATH)/net

HOSTCC          = gcc
HOSTCFLAGS      = -Wall -Wstrict-prototypes -O2 -fomit-frame-pointer

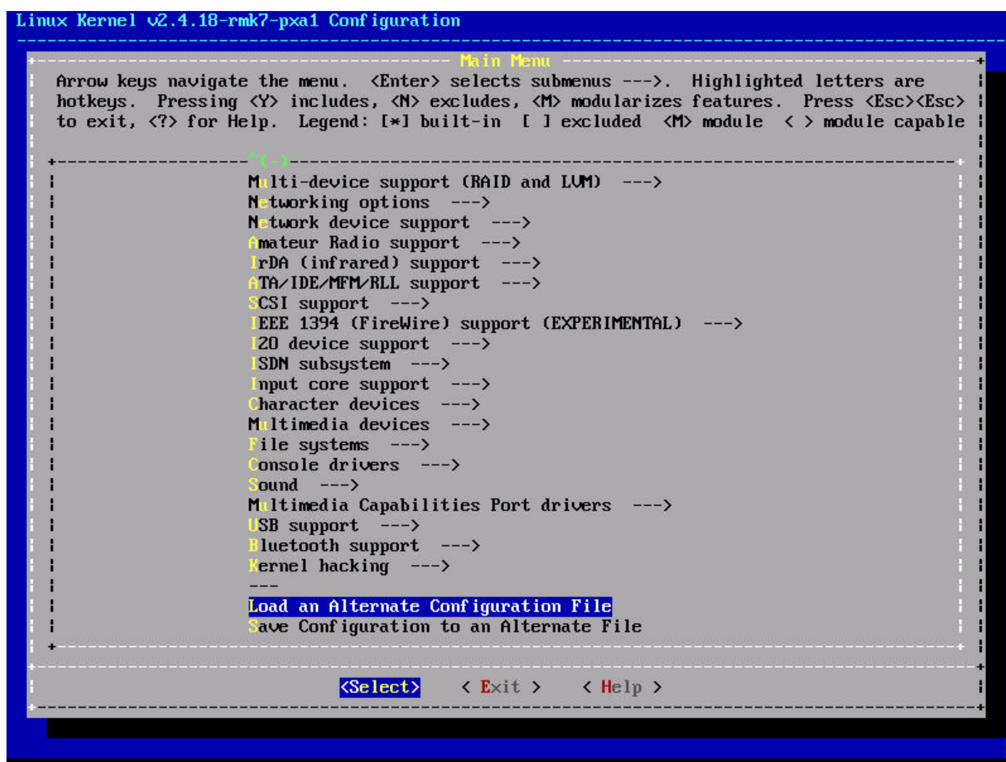
CROSS_COMPILE   = arm-linux-
#/opt/host/armv4l/bin/armv4l-unknown-linux-

#
# Include the make variables (CC, etc...)
#

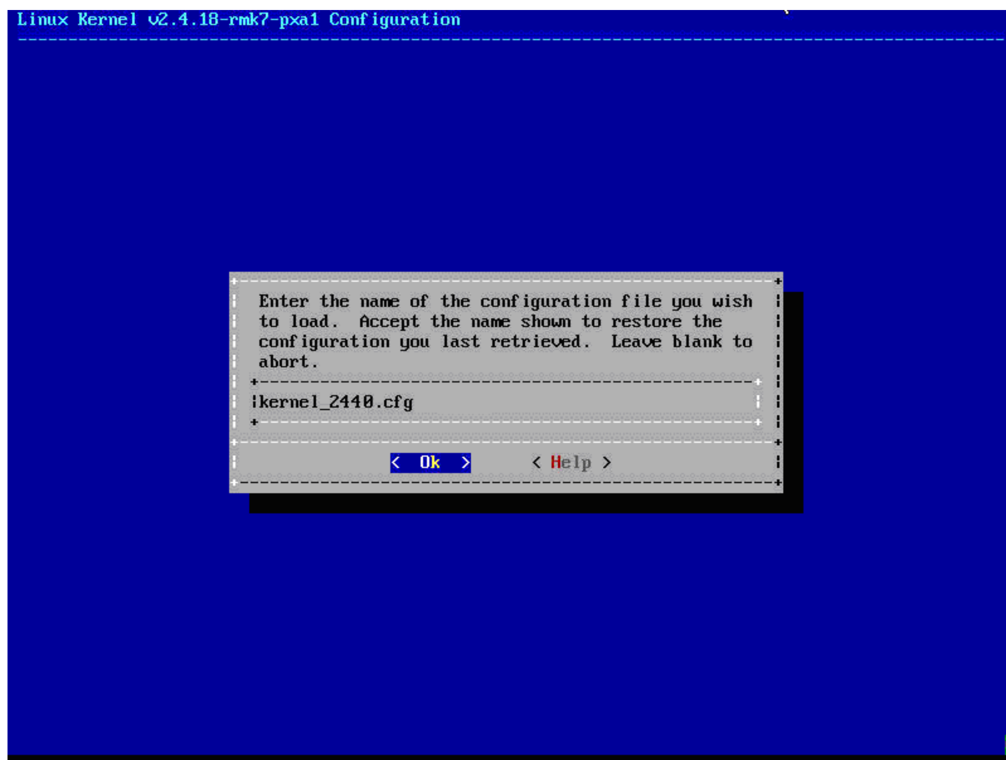
AS              = $(CROSS_COMPILE)as
LD              = $(CROSS_COMPILE)ld
CC              = $(CROSS_COMPILE)gcc
CPP             = $(CC) -E
AR              = $(CROSS_COMPILE)ar
NM              = $(CROSS_COMPILE)nm
STRIP           = $(CROSS_COMPILE)strip
:wq
```

6.4. 装载配置文件

执行输入命令：make menuconfig ，选择调入已有的配置文件

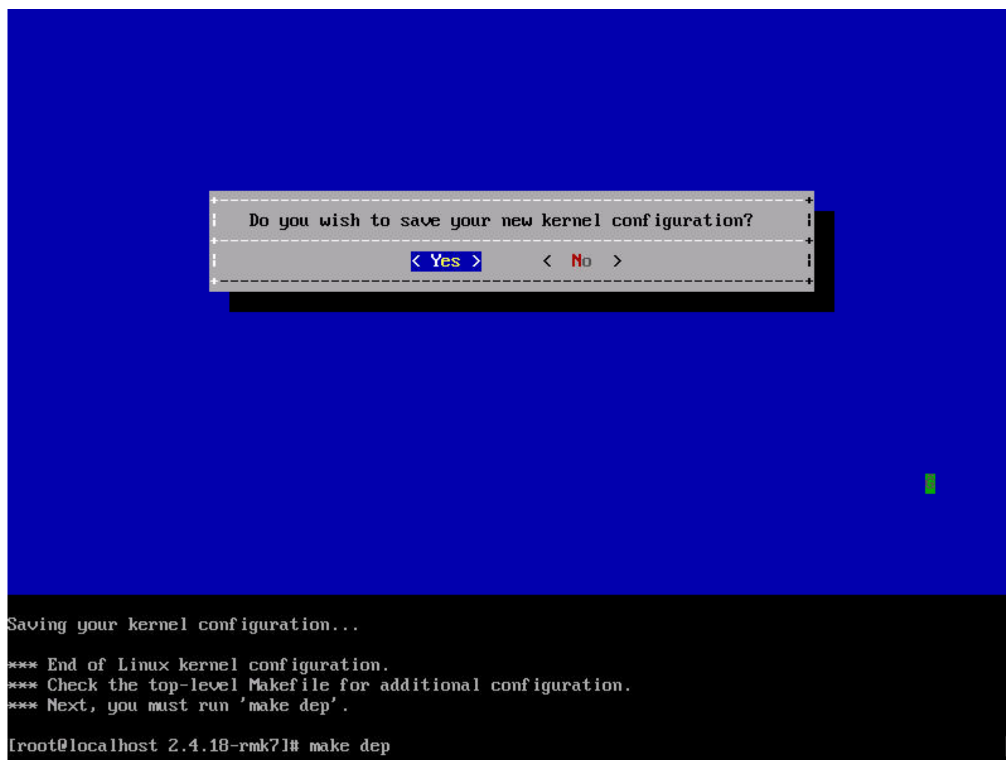


输入配置文件名并回车，在主菜单里选择 Exit 退出并保存设置。



6.5. 启动 MAKE DEP

执行 make dep


Do you wish to save your new kernel configuration?
< Yes >
< No >
Saving your kernel configuration...
*** End of Linux kernel configuration.
*** Check the top-level Makefile for additional configuration.
*** Next, you must run 'make dep'.
[root@localhost 2.4.18-rmk7]# make dep

```
/work/2.4.18-rmk7/scripts/mkdep -D__KERNEL__ -I/work/2.4.18-rmk7/include -Wall -Wstrict-prototypes -
Wno-trigraphs -Os -mapcs -fno-strict-aliasing -fno-common -fno-common -pipe -mapcs-32 -march=armv4 -
mtune=arm9tdmi -mshort-load-bytes -msoft-float  -- a3d.c adi.c amijoy.c analog.c cobra.c cs461x.c db
9.c emu10k1-gp.c gamecon.c gameport.c gf2k.c grip.c iforce.c interact.c lightning.c magellan.c ns558
.c pcigame.c serio.c serport.c sidewinder.c spaceball.c spaceorb.c stinger.c tmdc.c turbografx.c war
rior.c > .depend
make[6]: Leaving directory `/work/2.4.18-rmk7/drivers/char/joystick'
make -C mwave fastdep
make[6]: Entering directory `/work/2.4.18-rmk7/drivers/char/mwave'
/work/2.4.18-rmk7/scripts/mkdep -D__KERNEL__ -I/work/2.4.18-rmk7/include -Wall -Wstrict-prototypes -
Wno-trigraphs -Os -mapcs -fno-strict-aliasing -fno-common -fno-common -pipe -mapcs-32 -march=armv4 -
mtune=arm9tdmi -mshort-load-bytes -msoft-float -DMW_TRACE -- 3780i.c 3780i.h mwavedd.c mwavedd.h mwa
vepub.h smapi.c smapi.h tp3780i.c tp3780i.h > .depend
make[6]: Leaving directory `/work/2.4.18-rmk7/drivers/char/mwave'
make -C pcmcia fastdep
make[6]: Entering directory `/work/2.4.18-rmk7/drivers/char/pcmcia'
/work/2.4.18-rmk7/scripts/mkdep -D__KERNEL__ -I/work/2.4.18-rmk7/include -Wall -Wstrict-prototypes -
Wno-trigraphs -Os -mapcs -fno-strict-aliasing -fno-common -fno-common -pipe -mapcs-32 -march=armv4 -
mtune=arm9tdmi -mshort-load-bytes -msoft-float  -- memory_cs.c serial_cs.c > .depend
make[6]: Leaving directory `/work/2.4.18-rmk7/drivers/char/pcmcia'
make -C rio fastdep
make[6]: Entering directory `/work/2.4.18-rmk7/drivers/char/rio'
/work/2.4.18-rmk7/scripts/mkdep -D__KERNEL__ -I/work/2.4.18-rmk7/include -Wall -Wstrict-prototypes -
Wno-trigraphs -Os -mapcs -fno-strict-aliasing -fno-common -fno-common -pipe -mapcs-32 -march=armv4 -
mtune=arm9tdmi -mshort-load-bytes -msoft-float  -- board.h bootpkt.h brates.h cdproto.h chan.h cirru
s.h cmdblk.h cmd.h cmdpkt.h control.h daemon.h data.h debug.h defaults.h eisa.h enable.h error.h err
ors.h formpkt.h func.h host.h hosthw.h link.h linux_compat.h list.h lrt.h ltt.h lttwake.h map.h mca.
h mesg.h param.h parmmap.h pci.h phb.h pkt.h poll.h port.h proto.h protsts.h qbuf.h rioboard.h riobo
ot.c riocmd.c rioctrl.c riodrvr.h rio.h rioinfo.h rioinit.c riointr.c rioioctl.h rio_linux.c rio_lin
ux.h riolocks.h rioparam.c riopcicopy.c rioroute.c riospace.h riotable.c riotime.h riotty.c riotypes
.h riowinif.h riscos.h rom.h route.h rtahw.h rup.h rupstat.h sam.h selftest.h space.h sysmap.h timeo
uts.h top.h typdef.h unixrup.h > .depend
make[6]: Leaving directory `/work/2.4.18-rmk7/drivers/char/rio'
make[5]: Leaving directory `/work/2.4.18-rmk7/drivers/char'
make[4]: Leaving directory `/work/2.4.18-rmk7/drivers/char'
make -C dio fastdep
```

6.6. 编译 ZIMAGE

执行 make zImage

```
    trigraphs -Os -mapcs -fno-strict-aliasing -fno-common -fno-common -pipe -mapcs-32 -march=armv4 -
mtune=arm9tdmi -mshort-load-bytes -msoft-float  -- alignment.c consistent.c discontig.c extable.c fa
ult-armo.c fault-armv.c fault-common.c init.c ioremap.c mm-armo.c mm-armv.c mm-clps7500.c mm-ftvpci.
c mm-l7200.c mm-rpc.c mm-tbox.c proc-arm1020.S proc-arm2,3.S proc-arm6,7.S proc-arm720.S proc-arm920
.S proc-arm922.S proc-arm926.S proc-sa110.S proc-syms.c proc-xscale.S small_page.c > .depend
make[2]: Leaving directory `/work/2.4.18-rmk7/arch/arm/mm'
make -C arch/arm/lib fastdep
make[2]: Entering directory `/work/2.4.18-rmk7/arch/arm/lib'
/work/2.4.18-rmk7/scripts/mkdep -D__KERNEL__ -I/work/2.4.18-rmk7/include -Wall -Wstrict-prototypes -
Wno-trigraphs -Os -mapcs -fno-strict-aliasing -fno-common -fno-common -pipe -mapcs-32 -march=armv4 -
mtune=arm9tdmi -mshort-load-bytes -msoft-float  -- ashldi3.c ashrdi3.c backtrace.S changebit.S clear
bit.S copy_page.S csumipv6.S csumpartialcopygeneric.S csumpartialcopy.S csumpartialcopyuser.S csumpa
rtial.S delay.S ecard.S findbit.S floppydma.S gcclib.h getuser.S io-acorn.S io-readsb.S io-readsl-ar
mv3.S io-readsl-armv4.S io-readsw-armv3.S io-readsw-armv4.S io-shark.c io-writesb.S io-writesl.S io-
writesw-armv3.S io-writesw-armv4.S kbd.c lib1funcs.S longlong.h lshrdi3.c memchr.S memcpy.S memset.S
 memzero.S muldi3.c putuser.S setbit.S strchr.S strncpy_from_user.S strnlen_user.S strrchr.S testcha
ngebit.S testclearbit.S testsetbit.S uaccess-armo.S uaccess.S ucmpdi2.c udivdi3.c > .depend
make[2]: Leaving directory `/work/2.4.18-rmk7/arch/arm/lib'
make -C arch/arm/nwfpe fastdep
make[2]: Entering directory `/work/2.4.18-rmk7/arch/arm/nwfpe'
/work/2.4.18-rmk7/scripts/mkdep -D__KERNEL__ -I/work/2.4.18-rmk7/include -Wall -Wstrict-prototypes -
Wno-trigraphs -Os -mapcs -fno-strict-aliasing -fno-common -fno-common -pipe -mapcs-32 -march=armv4 -
mtune=arm9tdmi -mshort-load-bytes -msoft-float  -- ARM-gcc.h double_cpdo.c entry26.S entry.S extende
d_cpdo.c fpa11.c fpa11_cpdo.c fpa11_cpdt.c fpa11_cprt.c fpa11.h fpmodule.c fpmodule.h fpopcode.c fpo
pcode.h fpsr.h milieu.h single_cpdo.c softfloat.c softfloat.h > .depend
make[2]: Leaving directory `/work/2.4.18-rmk7/arch/arm/nwfpe'
make -C arch/arm/fastfpe fastdep
make[2]: Entering directory `/work/2.4.18-rmk7/arch/arm/fastfpe'
/work/2.4.18-rmk7/scripts/mkdep -D__KERNEL__ -I/work/2.4.18-rmk7/include -Wall -Wstrict-prototypes -
Wno-trigraphs -Os -mapcs -fno-strict-aliasing -fno-common -fno-common -pipe -mapcs-32 -march=armv4 -
mtune=arm9tdmi -mshort-load-bytes -msoft-float  -- CPDO.S CPDT.S CPRT.S entry.S module.c > .depend
make[2]: Leaving directory `/work/2.4.18-rmk7/arch/arm/fastfpe'
make[1]: Leaving directory `/work/2.4.18-rmk7'
[root@localhost 2.4.18-rmk7]# make zImage
gcc -Wall -Wstrict-prototypes -O2 -fomit-frame-pointer -o scripts/split-include scripts/split-includ
e.c
```

6.7. 获取内核压缩映像

编译结束后，可在 arch/arm/boot 目录下得到 LINUX 内核压缩映像 ZIMAGE。

```
make[1]: Leaving directory `/work/2.4.18-rmk7/arch/arm/fastfpe'
arm-linux-ld -p -X -T arch/arm/vmlinux.lds arch/arm/kernel/head-armv.o arch/arm/kernel/init_task.o i
nit/main.o init/version.o \
        --start-group \
        arch/arm/kernel/kernel.o arch/arm/mm/mm.o arch/arm/mach-s3c2410/s3c2410.o kernel/kernel.o mm
/mm.o fs/fs.o ipc/ipc.o \
         drivers/serial/serial.o drivers/char/char.o drivers/block/block.o drivers/misc/misc.o drive
rs/net/net.o drivers/media/media.o drivers/scsi/scsidrv.o drivers/sound/sounddrivers.o drivers/mtd/m
tdlink.o drivers/net/wireless/wireless_net.o drivers/video/video.o drivers/usb/usbdrv.o drivers/inpu
t/inputdrv.o \
        net/network.o \
        arch/arm/nwfpe/math-emu.o arch/arm/lib/lib.a /work/2.4.18-rmk7/lib/lib.a \
        --end-group \
        -o vmlinux
arm-linux-nm vmlinux | grep -v '\(compiled\)\|\(\.o$\)\|\( [aUw] \)\|\(\.\.ng$\)\|\(LASH[RL]DI\)' |
sort > System.map
make[1]: Entering directory `/work/2.4.18-rmk7/arch/arm/boot'
make[2]: Entering directory `/work/2.4.18-rmk7/arch/arm/boot/compressed'
arm-linux-gcc -D__ASSEMBLY__ -D__KERNEL__ -I/work/2.4.18-rmk7/include -mapcs-32 -march=armv4 -mno-fp
u -msoft-float -traditional -c head.S
arm-linux-gcc -D__KERNEL__ -I/work/2.4.18-rmk7/include -O2 -DSTDC_HEADERS -mapcs-32 -march=armv4 -mt
une=arm9tdmi -mshort-load-bytes -msoft-float -fpic -D__KERNEL__ -I/work/2.4.18-rmk7/include  -c -o m
isc.o misc.c
arm-linux-gcc -D__ASSEMBLY__ -D__KERNEL__ -I/work/2.4.18-rmk7/include -mapcs-32 -march=armv4 -mno-fp
u -msoft-float   -c -o head-s3c2410.o head-s3c2410.S
arm-linux-objcopy -O binary -R .note -R .comment -S /work/2.4.18-rmk7/vmlinux piggy
gzip -9 < piggy > piggy.gz
arm-linux-ld -r -o piggy.o -b binary piggy.gz
rm -f piggy piggy.gz
arm-linux-ld -p -X -T vmlinux.lds head.o misc.o head-s3c2410.o piggy.o /usr/local/arm/2.95.3/lib/gcc
-lib/arm-linux/2.95.3/libgcc.a -o vmlinux
make[2]: Leaving directory `/work/2.4.18-rmk7/arch/arm/boot/compressed'
arm-linux-objcopy -O binary -R .note -R .comment -S compressed/vmlinux zImage
make[1]: Leaving directory `/work/2.4.18-rmk7/arch/arm/boot'
[root@localhost 2.4.18-rmk7]# ls arch/arm/boot/zImage -l
-rwxr-xr-x    1 root     root       817532 May  6 21:45 arch/arm/boot/zImage
[root@localhost 2.4.18-rmk7]#
```

生成内核映像后可以尝试烧写测试一下是否能运行，值得注意的是测试内核的前提是烧好了根文件系统，单独烧写内核映像是不能工作的。你可以用本公司光盘提供的现有的根文件系统进行测试，也可以自已做根文件系统。

6.8. cramfs 根文件系统的制作

这里我们以向 YL2440 光盘附带的 YL2440_camare_demo.cramfs 根文件系统的添加为例。

（1） 将 YL2440_camare_demo.cramfs 拷贝到任意目录下

（2） 在该目录下建立两个文件：

mkdir roms

mkdir tmp

（3） 将 YL2440_camare_demo.cramfs 挂接到 chang 目录

mount YL2440_camare_demo.cramfs roms -o loop

（4） 将 roms 目录下的内容压缩

tar -cvf 1.tar roms

这样将在 roms 的上一级目录产生一个 1.tar 的包

```
Telnet 192.168.0.112
[root@linuxserver zzmworkspace]#
[root@linuxserver zzmworkspace]#
[root@linuxserver zzmworkspace]#
[root@linuxserver zzmworkspace]#
[root@linuxserver zzmworkspace]#
[root@linuxserver zzmworkspace]#
[root@linuxserver zzmworkspace]#
[root@linuxserver zzmworkspace]#
[root@linuxserver zzmworkspace]#
[root@linuxserver zzmworkspace]#
[root@linuxserver zzmworkspace]#
[root@linuxserver zzmworkspace]#
[root@linuxserver zzmworkspace]#
[root@linuxserver zzmworkspace]#
[root@linuxserver zzmworkspace]#
[root@linuxserver zzmworkspace]#
[root@linuxserver zzmworkspace]# ls
hello.c  s3c2440_kernel2.4.18_rel         tmp
roms     s3c2440_kernel2.4.18.tar.bz2     YL2440_camare_demo.cramfs
[root@linuxserver zzmworkspace]# mount YL2440_camare_demo.cramfs roms -o loop
[root@linuxserver zzmworkspace]# cd roms
[root@linuxserver roms]# ls
bin  dev  etc  home  lib  linuxrc  proc  Qtopia  ramdisk  sbin  tmp  usr  var
[root@linuxserver roms]# cd ..
[root@linuxserver zzmworkspace]# tar -cvf 1.tar roms
```

建立压缩文件

（5） 将包解压到 tmp 目录下。

umount roms ； 卸载挂接

cd .. ； 进入上一级目录

mv 1.tar tmp ；

cd tmp ；

tar –xvf 1.tar ；将打包的根文件系统的里的内容解压

rm 1.tar

（6）经过上面的步骤就可以将自己的驱动和应用程序添加到 cramfs 根文件系统中了

现在将开始制作 cramfs 根文件系统

你可以进入 tmp 目录下的文件夹(例如 roms),用 ls 命令看一看是不是有下图这样一些文件,

```
Telnet 192.168.0.112
[root@linuxserver roms]# ls
bin  etc   lib      proc    ramdisk  tmp  var
dev  home  linuxrc  Qtopia  sbin     usr  zzm
[root@linuxserver roms]# rm -drf zzm
[root@linuxserver roms]# ls
bin  dev  etc  home  lib  linuxrc  proc  Qtopia  ramdisk  sbin  tmp  usr  var
[root@linuxserver roms]# ls
bin  dev  etc  home  lib  linuxrc  proc  Qtopia  ramdisk  sbin  tmp  usr  var
[root@linuxserver roms]#
[root@linuxserver roms]#
[root@linuxserver roms]#
[root@linuxserver roms]#
[root@linuxserver roms]#
[root@linuxserver roms]#
[root@linuxserver roms]#
[root@linuxserver roms]# ls
bin  dev  etc  home  lib  linuxrc  proc  Qtopia  ramdisk  sbin  tmp  usr  var
[root@linuxserver roms]# mkdir ucdragon
[root@linuxserver roms]# ls
bin  etc   lib      proc    ramdisk  tmp       usr
dev  home  linuxrc  Qtopia  sbin     ucdragon  var
[root@linuxserver roms]# cd ..
[root@linuxserver tmp]# ls
1.tar  hello.c  roms  tmps  YL240.cramfs  YL2440A.cramfs  YL2440.cramfs
[root@linuxserver tmp]# mkcramfs roms YL2440A_camare.cramfs_
```

自定义文件夹进行测试

建立根文件系

在这个目录下运行命令：

mkcramfs guo YL2440_camare.cramfs

运行成功后，会在该目录下生成 YL2440_camare.cramfs 根文件系统

（7）根文件系统制作成功后，就可以将 YL2440_camare.cramfs 烧写到相应的地方，关于

根文件系统的烧写，在“YL2440 使用手册”中有如何烧写根文件系统的说明。

（8）上面的步骤教你如何将自己的驱动和引用程序添加到根文件系统中。

烧好内核的开发板中再将现在做的根目录文件系统烧写进去，设好相应的启动配置，就能看看烧入的文件系统是否有效了。

第七章 YL2440 开发系统 WINCE4.2 使用手册

这一章的工作平台是：2000/xp。

7.1.WINCE 的安装

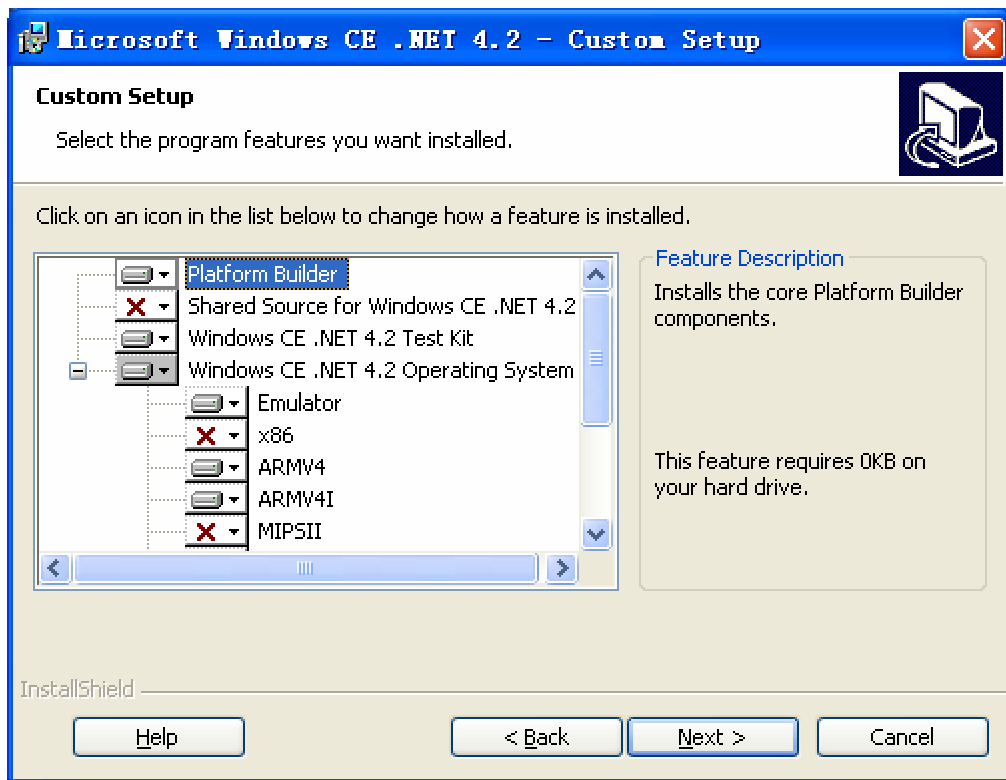
7.1.1.安装开发环境

为了建立WINCE4.2的应用开发环境，您需要准备好由微软公司发布的两张PB（Platform Builder 4.2）安装光盘（即为本公司的一张PB4.2的DVD安装盘），这两张光盘分别被命名为PB4-A, PB4-B。这两张光盘包含了微软公司的Windows® CE .NET 4.2 操作系统安装程序，以及把操作系统编译移植到指定目标硬件平台的工具－平台建立器（Platform Builder 4.2）。

下面介绍如何从这两张光盘安装WINCE4.2操作系统和Platform Builder 4.2。首先将两张光盘的内容全部拷贝到同一个目录下，例如：D:\PB4SET

运行SETUP.EXE, 输入串号

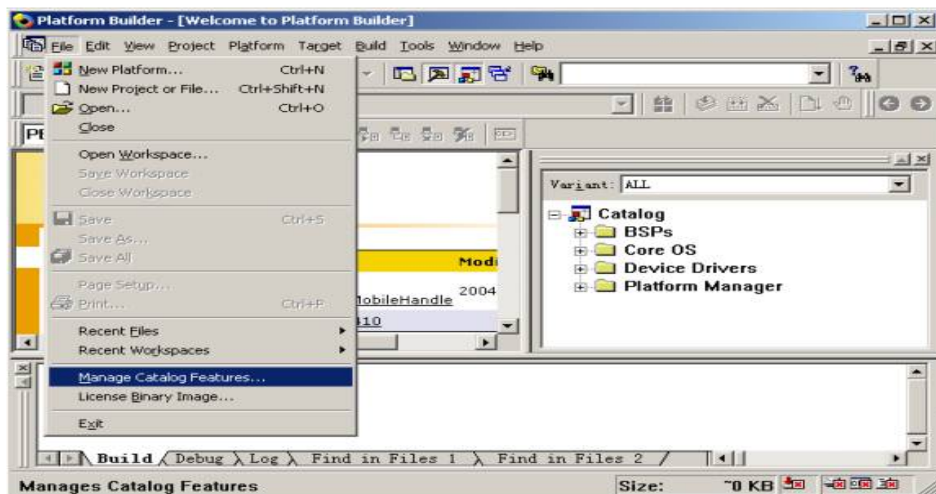
在出现安装选项时选择ARMV4,ARMV4I(如图一)



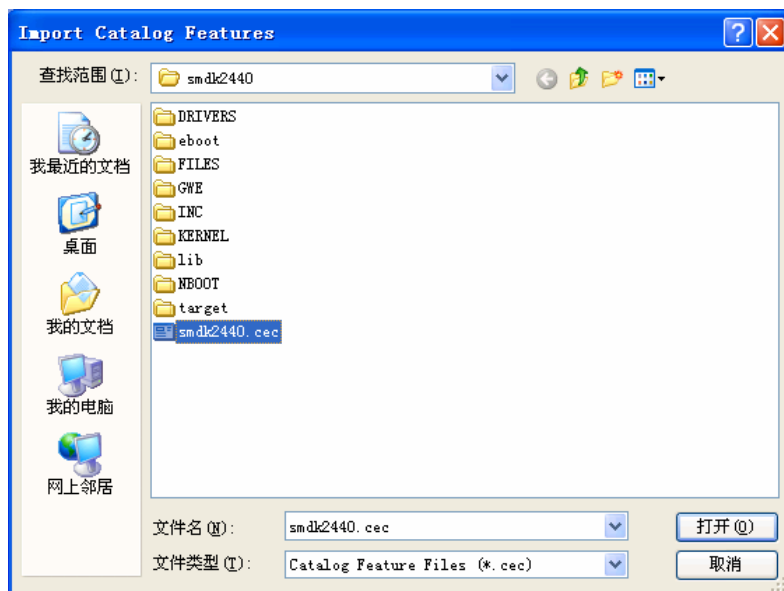
安装结束时，WINCE目录有3.5G左右的文件。

7.1.2. 安装目录

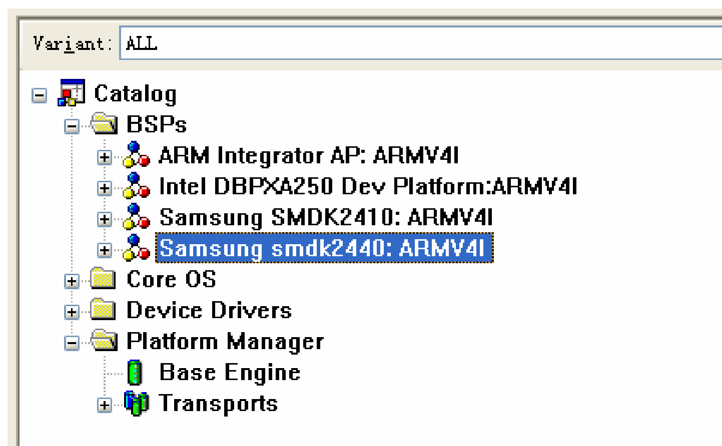
前面已经安装了平台建立器Platform Builder 4.2，下一步，要运行Platform Builder，并对它进行一系列的设置，目的是为编译WINCE4.2操作系统映像做好准备。点击PB主菜单“File”下的“Manage Catalog Features”菜单项，如下图：



您需要“导入”（Import）新的“smdk2440.cec”文件。新的“smdk2440.cec”文件是从2440用户光盘的wince目录中的smdk2440.rar压缩包解压出来的，一般我们将这个压缩包解压缩到wince420的安装目录下的PLATFORM 子目录，例如WINCE420\PLATFORM；然后就可以点击“Import” 按钮导入新的“SMDK2440.CEC”文件了，打开WINCE420\PLATFORM\smdk2440目录，选中该目录下的“smdk2440.cec”文件，如下图：

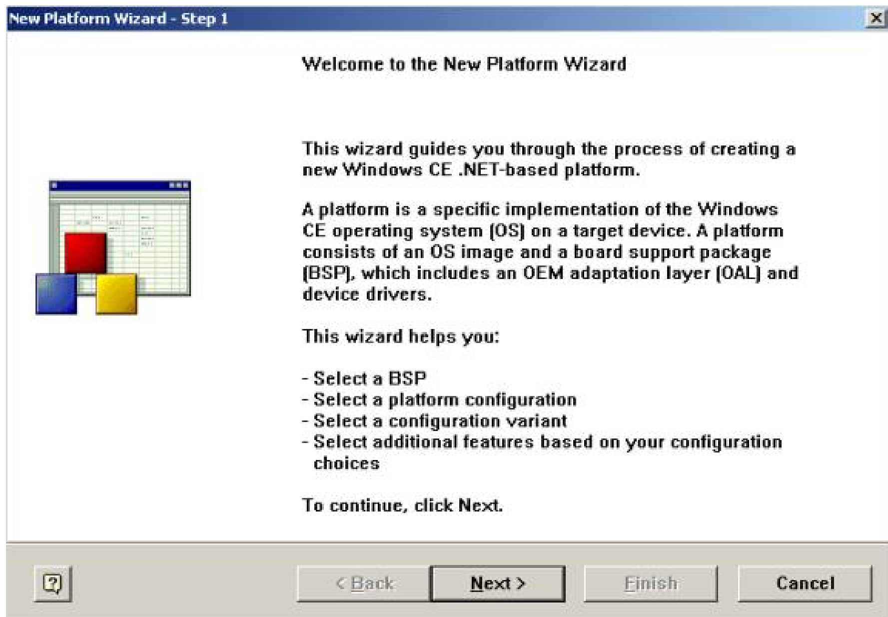


导入之后，在PB的目录查看器上将会看到“Samsung smdk2440:ARMV4I”列，如下图：

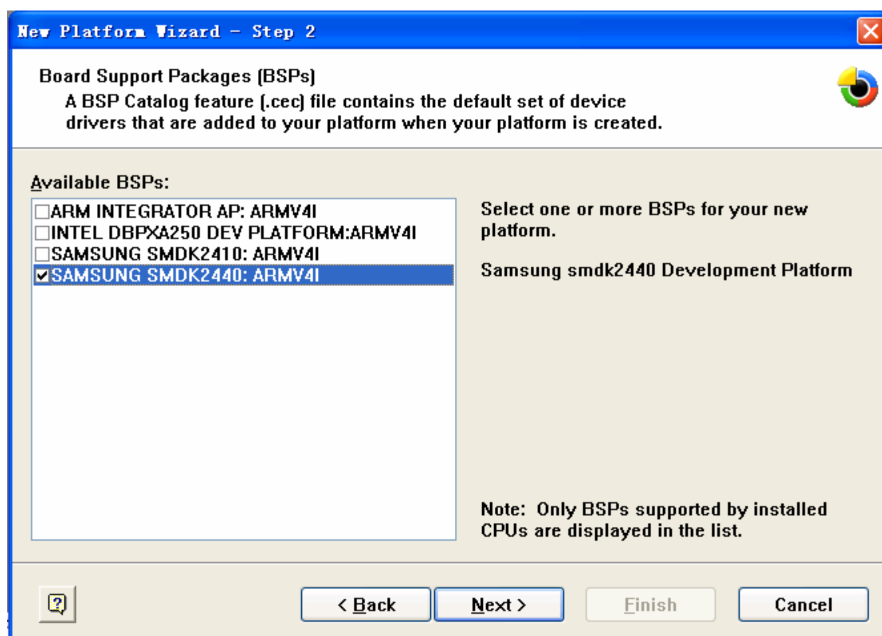


7.1.3.构建新的平台、编译

点击PB主菜单“File”下的“New Platform”菜单项，
将会出现“New Platform Wizard - Step 1”框，点击“Next” 按钮：

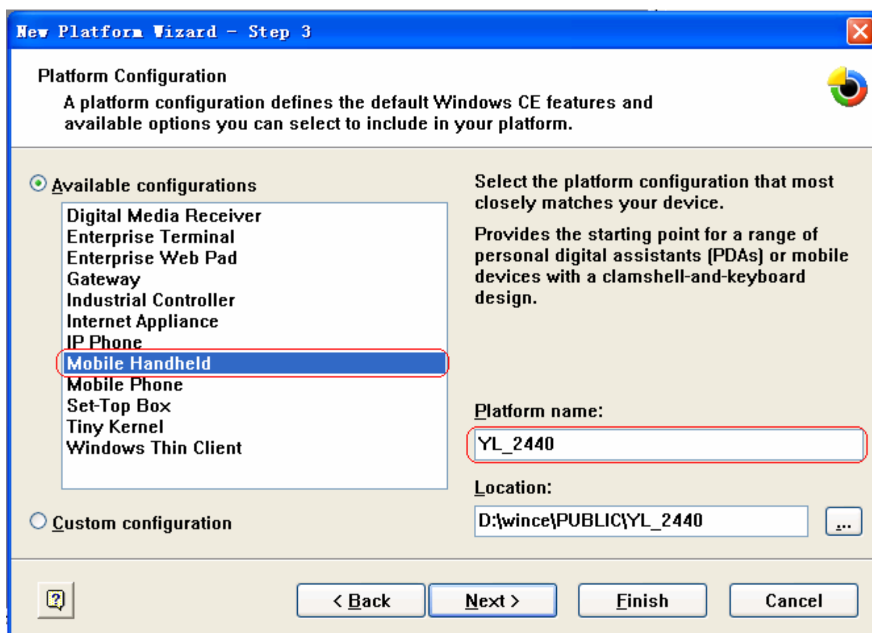


接着出现“New Platform Wizard - Step 2”框，您可看到“SAMSUNG SMDK2440:ARMV4I” 的BSP。点击“SAMSUNG SMDK2440:ARMV4I” 、再点击“Next”按钮，如下图：



出现“New Platform Wizard – Step 3”对话框，在“Available configurations”列表中选择您希望的配置，在”Platform name:”

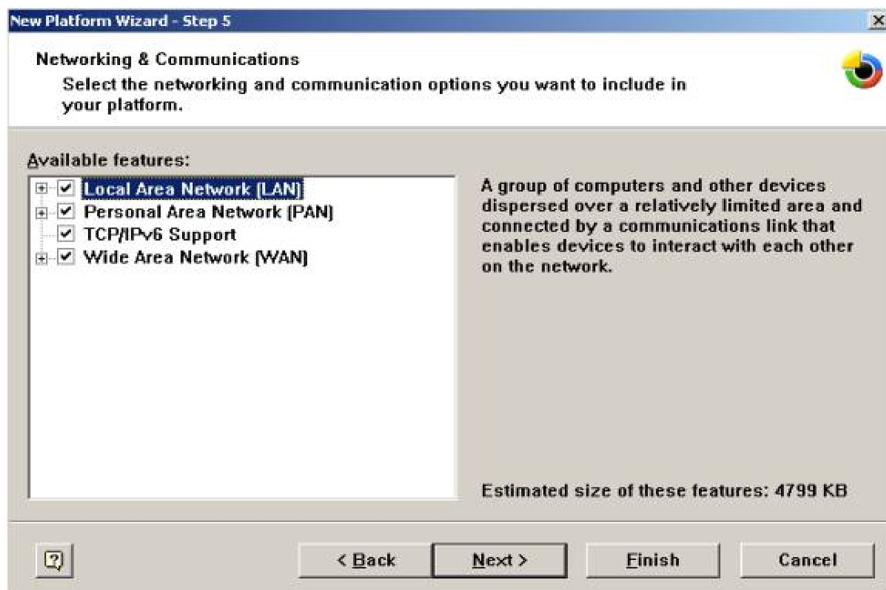
框中输入平台的名称YL_2440(可以输入其他的名字)、再点击“Next”按扭，如下图：



接着出现“New Platform Wizard - Step 4”框，选择您需要的“Application & Media”，如下图：



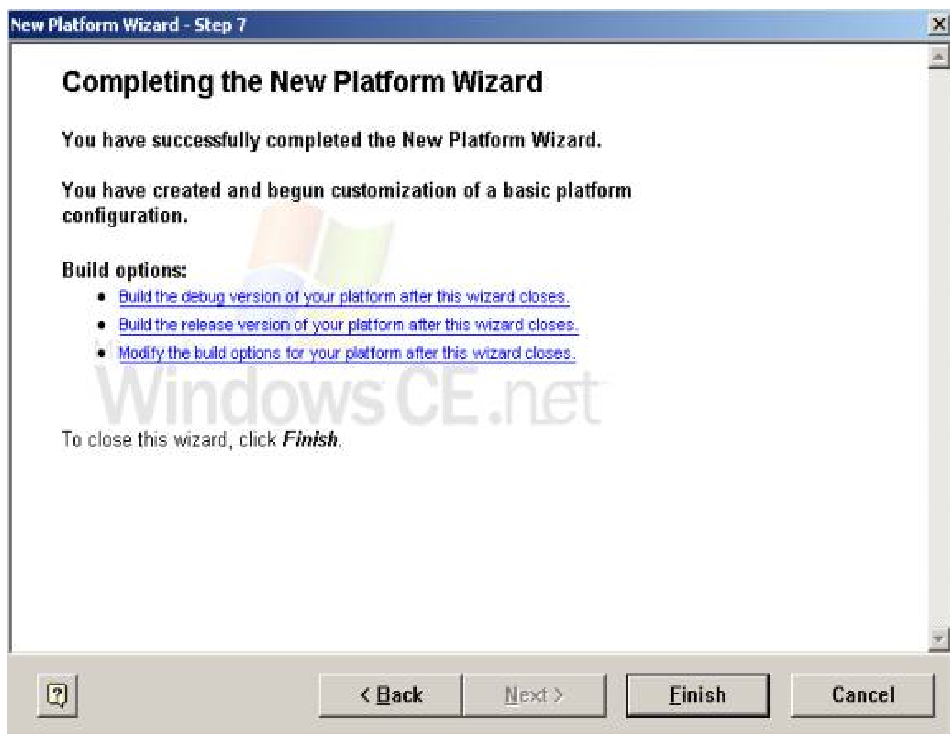
接着出现“New Platform Wizard－Step 5”框，选择您需要的“Networking & Communications”，如下图：



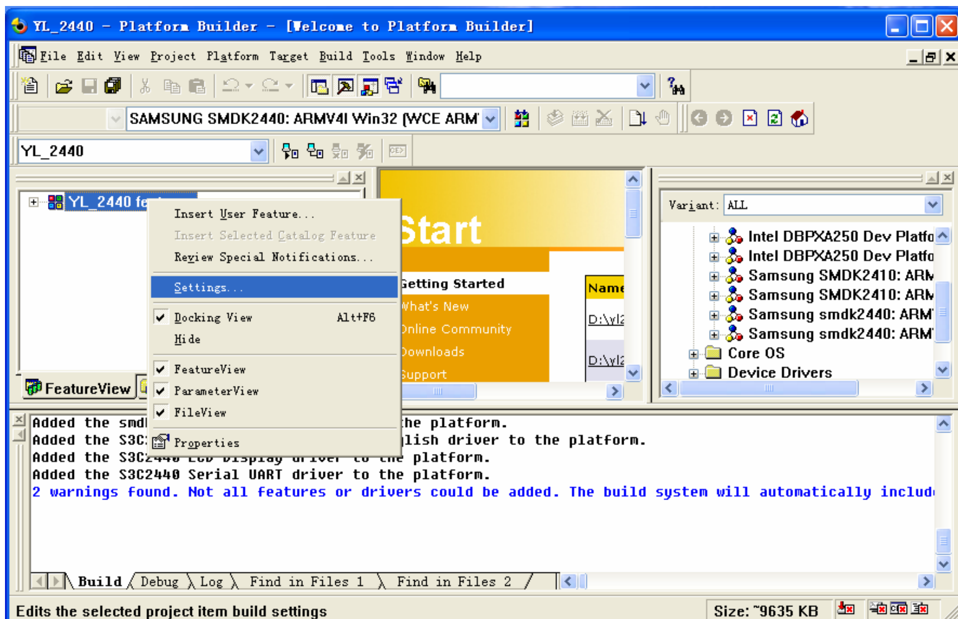
再点击”Next”按钮，出现以下的对话框。



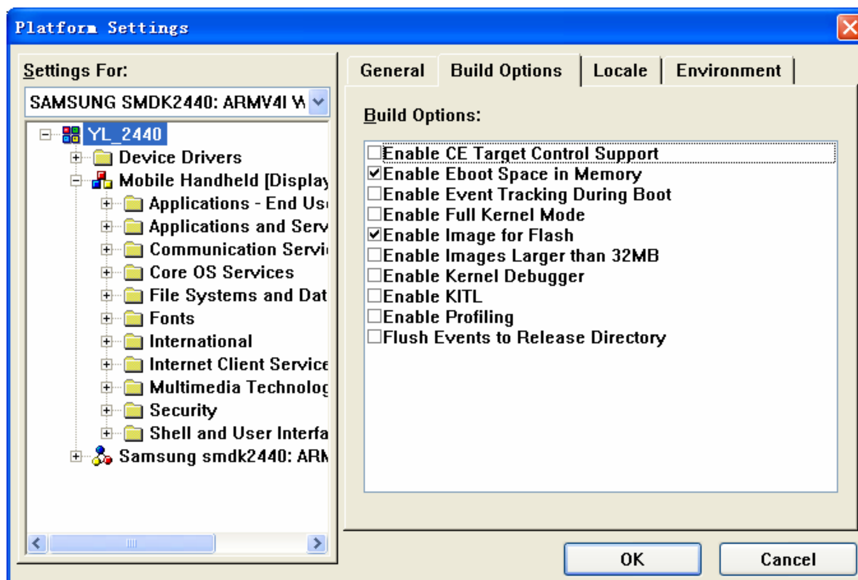
再点击“Next”按钮，您将看到“Completing the New Platform Wizard”对话框。建立新平台的所有设置步骤已经完成了。请点击“Finish”按钮。



下一步设置平台，点击 PB 的 Platform| Setting 菜单，如下图：

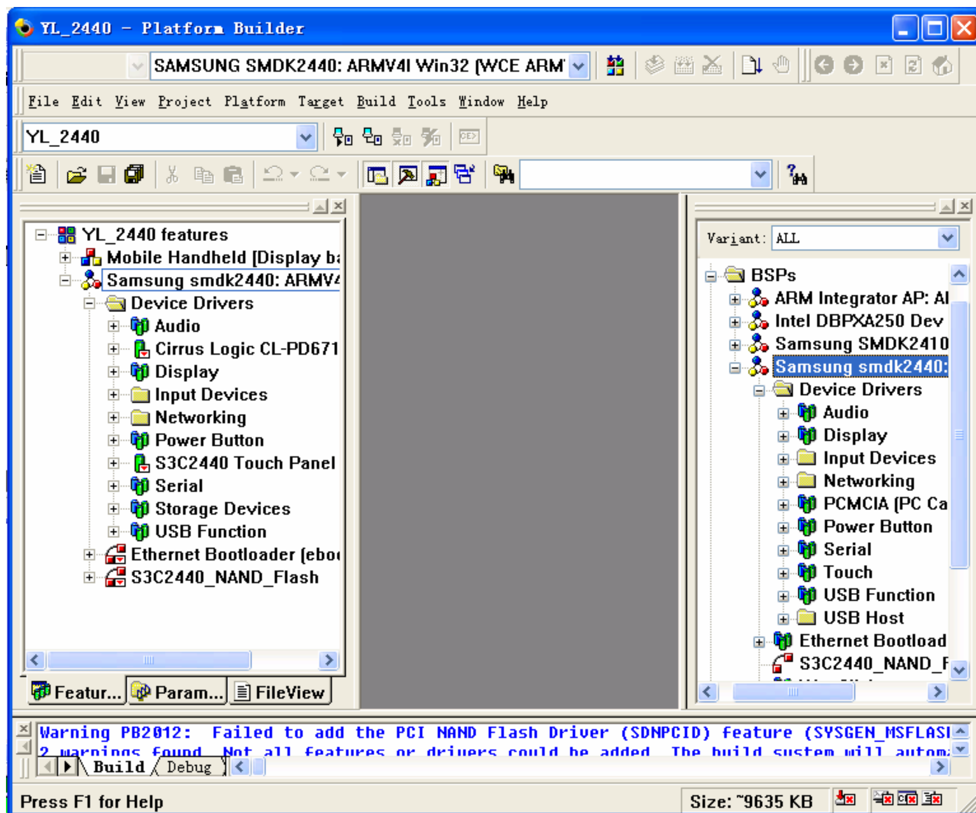


在弹出的“Platform Settings”设置框的“Build Options”标签页，配置如下：



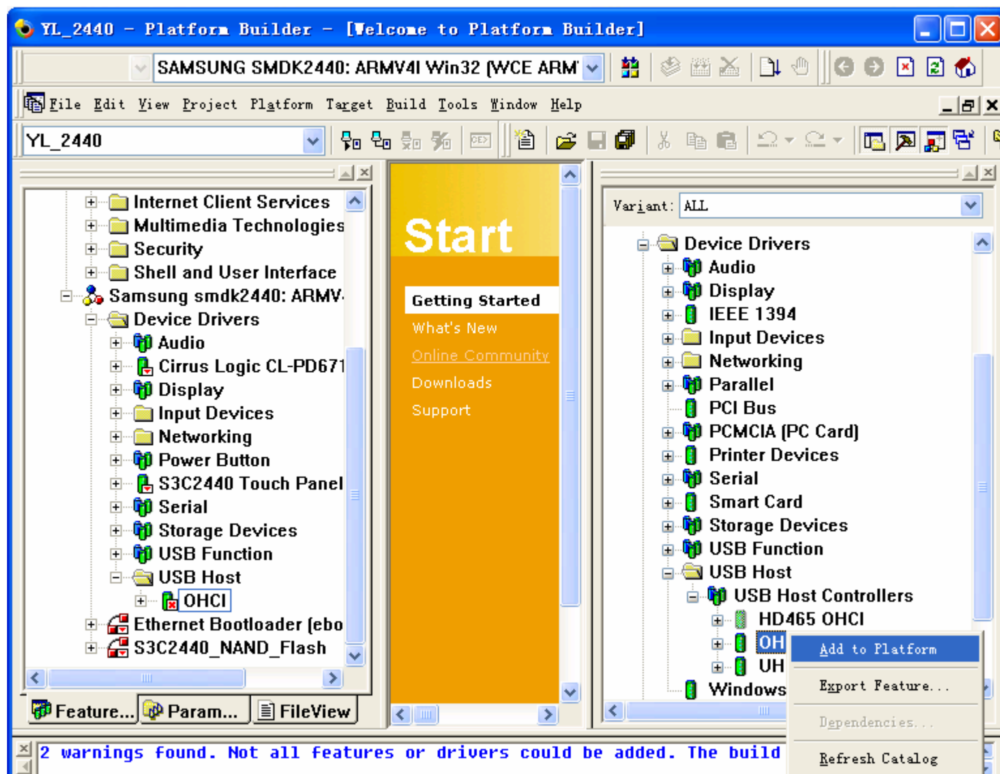
7.1.4. 添加驱动

展开右边的 Catalog 列表，如下图：



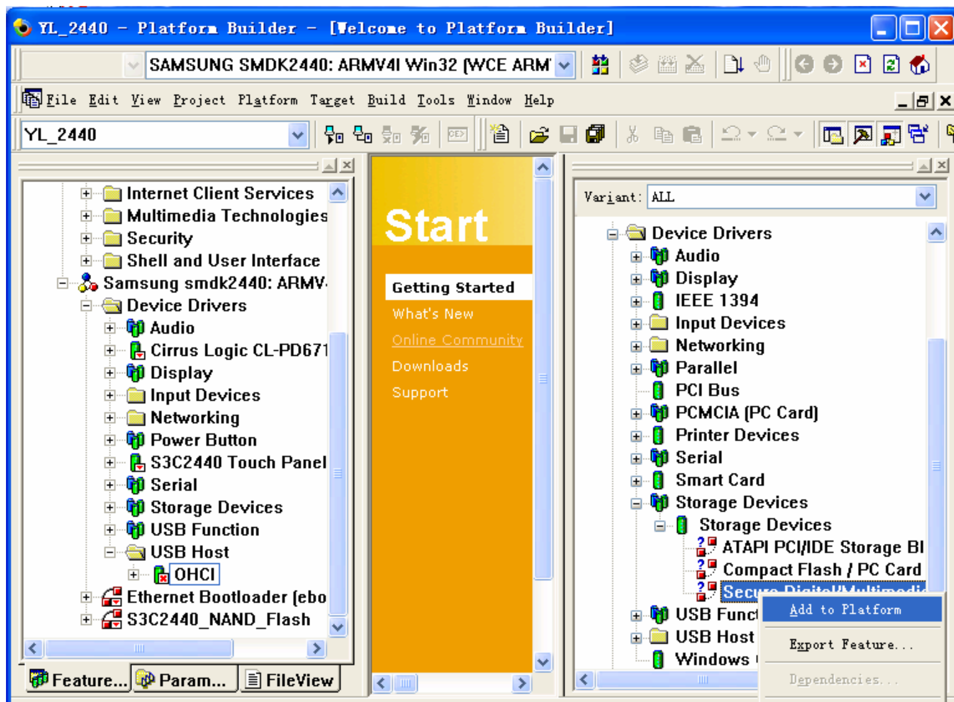
（1） 添加 USB HOST 驱动

在 Catalog 列表中 Device Drivers->USB Host ->USB Host Controllers->OCHI



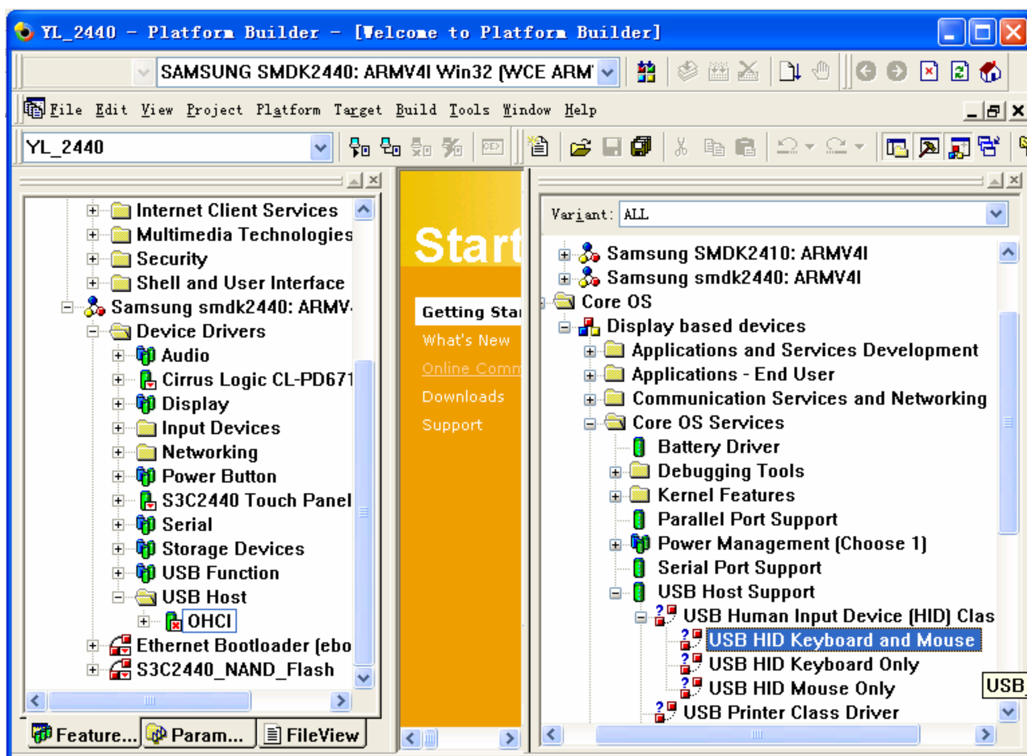
（2）添加 SD 卡驱动

在 Catalog 列表中 Device Drivers->Storage Devices->Secure Digital/Multimedia Card[SD/MMC]栏目。



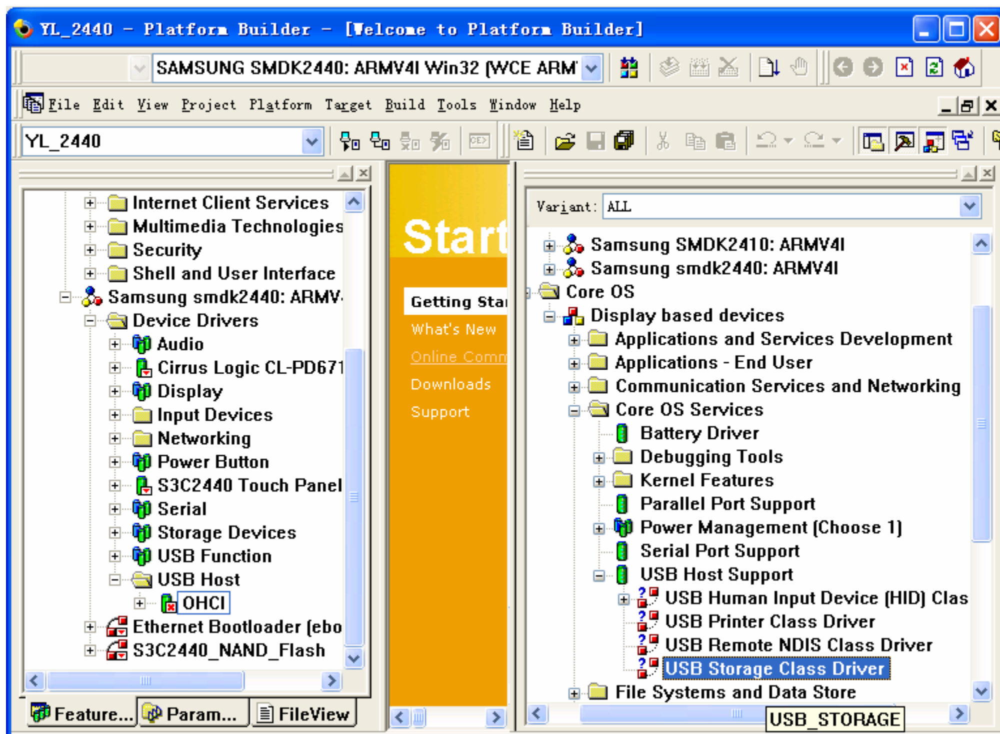
（3）添加鼠标和键盘驱动

在 Catalog 列表中 Core OS->Display and Services Development—>Core OS Services->USB Host Support ->USB Human Input Device[HID] ClassDriver 下的 USB HID Keyboard and Mouse 组件上单击鼠标右键，在右键菜单中选择 Add to Platform,将该组件加入到当前平台，如下图：

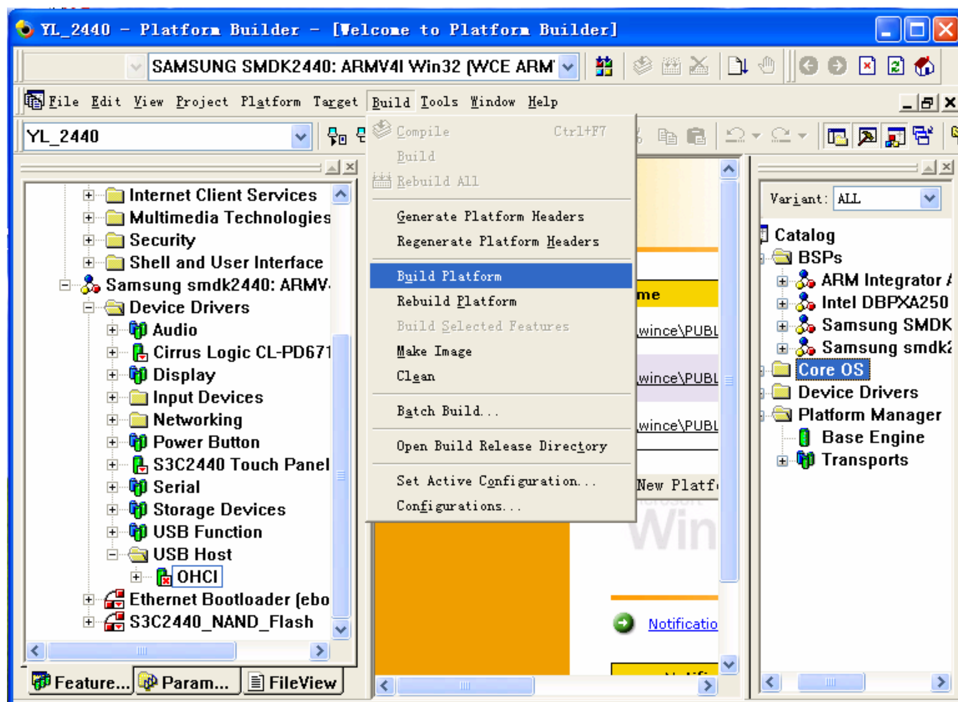


（4）添加 U 盘驱动

在 Catalog 列表中 Core OS->Display based devices->Core OS Services->USB Host Support->USB Storage Class Driver 组件上单击鼠标右键，在右键菜单中选择 Add to Platform，将该组件加入到当前平台，如下图：



驱动和设置完成后，点击 Build | Build Platform 开始编译平台



编译完成后，您就有了SMDK2440 的二进制的image：“nk.bin”和“nk.nb0”，一般而言，这两个文件位于编译平台时生成的文件夹“WINCE420\PUBLIC\[PlatformName]\RelDir\SAMSUNG_SMDK2440_ARMV4Release”。

7.2.WINCE 的 image 的运行

前面编译平台生成的文件nk.nb0，就是wince4.2操作系统以2440硬件平台为目标而编译生成的二进制映像文件（image）。这个文件可以被下载到YL2440的RAM空间执行，执行的结果就是在RAM中直接运行了wince4.2操作系统，这种方式并不会把wince映像固化到2440的FLASH中，系统调电之后在RAM中的wince就不复存在

了。可以通过以下两种方式把nk.nb0映像文件下载到YL2440的RAM空间执行。

关于WINCE的IMAGE在RAM空间运行，请看第四章节。

7.3. WINCE 的烧写

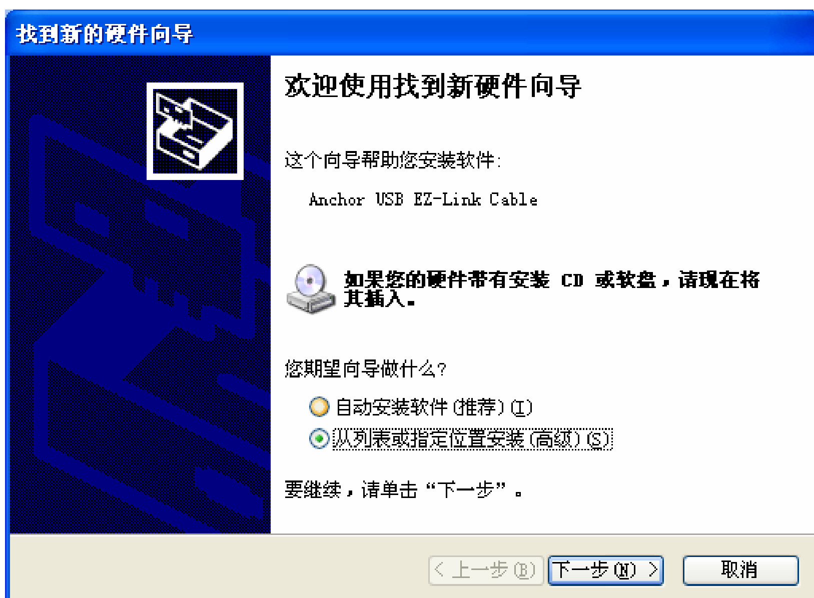
WINCE的nk.nb0 的烧写，请看YL2440 的使用手册第四章节。

7.4. 在 WINCE 和桌面系统之间建立通讯连接

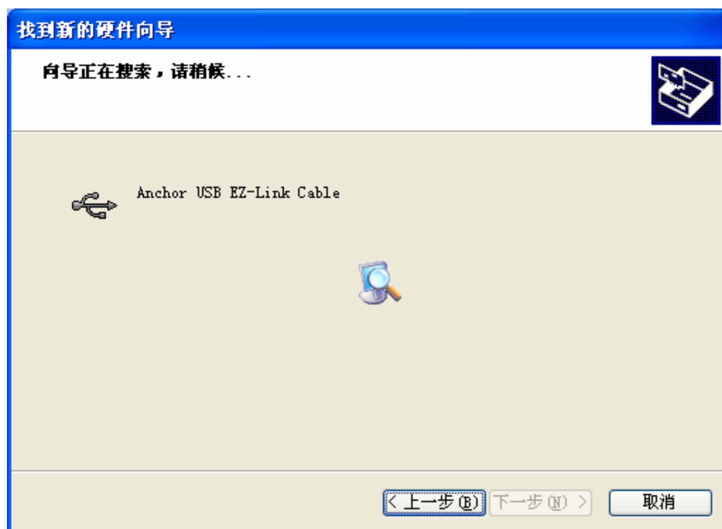
可以使用微软公司提供的一些工具，并在进行了一些必要的设置后，就可以在安装了 WINCE 操作系统的移动设备和 windows 桌面系统之间进行通讯连接，从而可以实现文件上传下载，远程调试等功能。

7.4.1. 安装驱动

启动 WINCE 后，用 USB 线连好 USB DEIVCE 和 PC 的 USB 端口，如果以前没有安装 WINCE 下的驱动，这时插上 USB 线后，在计算机端会出现“发现新硬件”的提示，这时就需要安装驱动了，驱动的位置在 wince 的 BSP 包里（smdk2440/DRIVERS/USB/FUNCTION），安装好 USB 驱动，就可以进行下面的操作了。


找到新的硬件向导
欢迎使用找到新硬件向导
这个向导帮助您安装软件:
Anchor USB EZ-Link Cable
如果您的硬件带有安装 CD 或软盘，请现在将其插入。
您期望向导做什么?
自动安装软件 (推荐) (I)
从列表或指定位置安装 (高级) (S)
要继续，请单击“下一步”。
< 上一步 (B)
下一步 (N) >
取消

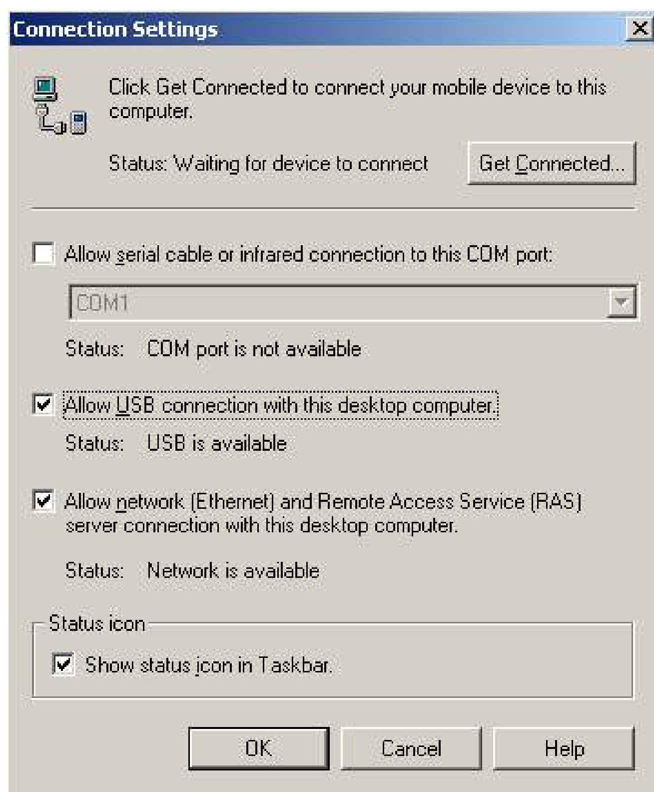

找到新的硬件向导
请选择您的搜索和安装选项。
在这些位置上搜索最佳驱动程序 (S)。
在搜索中包括这个位置 (O):
C:\WINCE420\PLATFORM\SMDK2440\D
不要搜索。我要自己选择要安装的驱动程序
浏览文件夹
选择包含您的硬件的驱动程序的文件夹。
DRIVERS
backlite
camera
CS8900
DISPLAY
DRVLIB
KEYBD
Nandflsh
PCMCIA
PWRBTN_PCF
serial
TOUCHP
USB
APP
FUNCTION
HCD
要查看任何子文件夹，请单击上面的 + 号。
确定
取消



7.4.2. 使用微软 ActiveSync 同步传输工具进行通讯连接

首先下载 ActiveSync 工具的安装程序 MSASYNC.EXE（这个工具可以单独从网上下载），安装，运行。

然后开始对 ActiveSync 进行设置，点击菜单 File | Connection Settings ，在弹出的设置对话框中，选择允许通过 USB 口、以太网口进行通讯连接，串口就不用选上了，以免同 DNW 冲突。如下图所示



ActiveSync 可以通过串口，USB 口，网口等方式建立连接，下面仅以 USB 连接方式为例介绍如何通过 ActiveSync 在桌面系统与 WINCE 之间建立通讯连接。

第一步，在确认桌面系统的 USB 口和 YL2440 的方形 USB 口已经通过 USB 线缆连接起来后，开始按以下步骤设置 YL2440 上运行的 wince 操作系统：

打开【我的电脑】—》打开【控制面板】—》打开【网络和拨号连接】—》点击【新建连接】，在“新建连接”设置对话框中，选择连接类型为“直接连接”，如下图



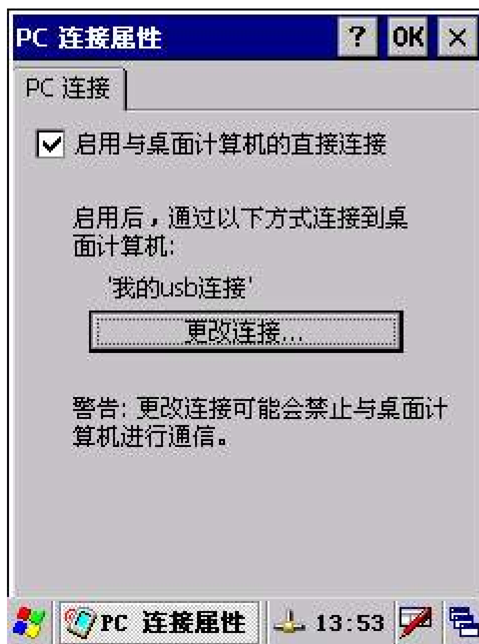
改名输入“我的 usb 连接”，点击“下一步”，在出现的“选择设备”下拉列表中选择“S3C2440 USB Cable:”，如下图



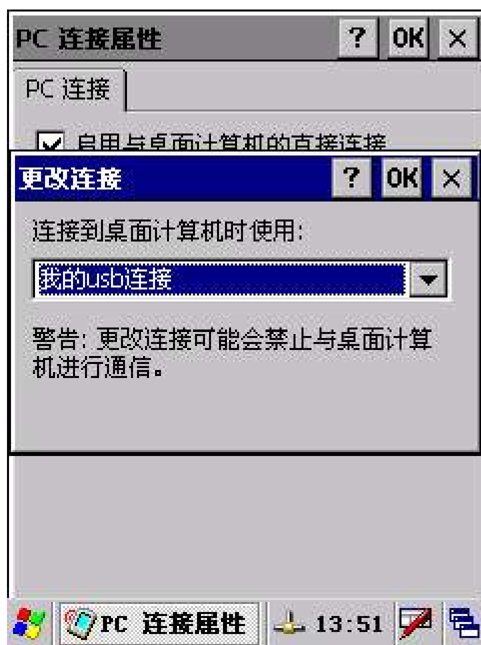
点击完成，这时就出现“我的 USB 连接”图标，如下图。



回到“控制面板”，点击【PC 连接】图标，进入“PC 连接属性”设置对话框，选中“启用与桌面计算机的直接连接”的复选框，然后再点击“更改连接”按钮，如下图，

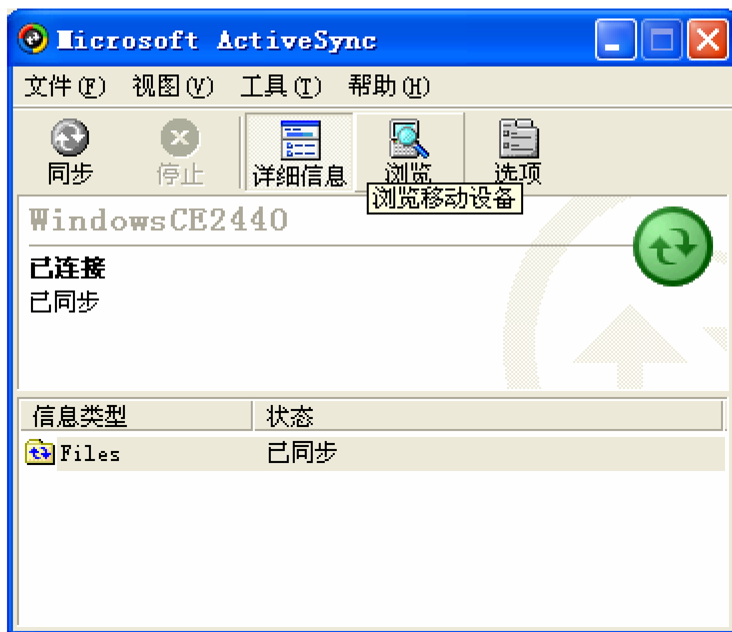


在“更改连接”设置对话框的下拉列表框中选择刚才新建的连接“我的 USB 连接”，然后按“OK”键退出。这样 WINCE 这边的设置宣告结束。

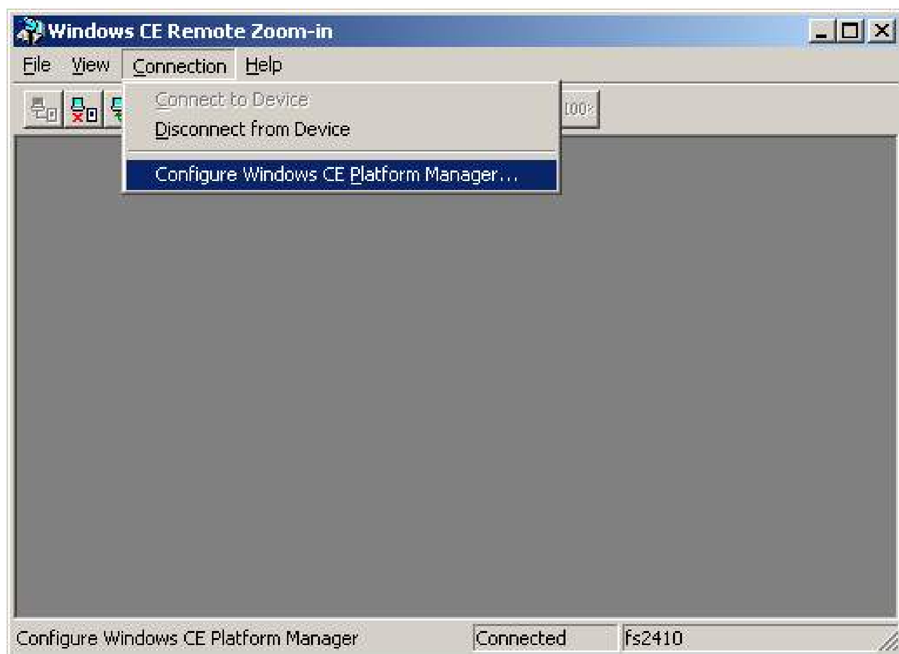


最后，回到 PC 机上来，打开 ActiveSync 的 File | Get Connected 菜单项，开始尝试和 WINCE 操作系统进行连接…….

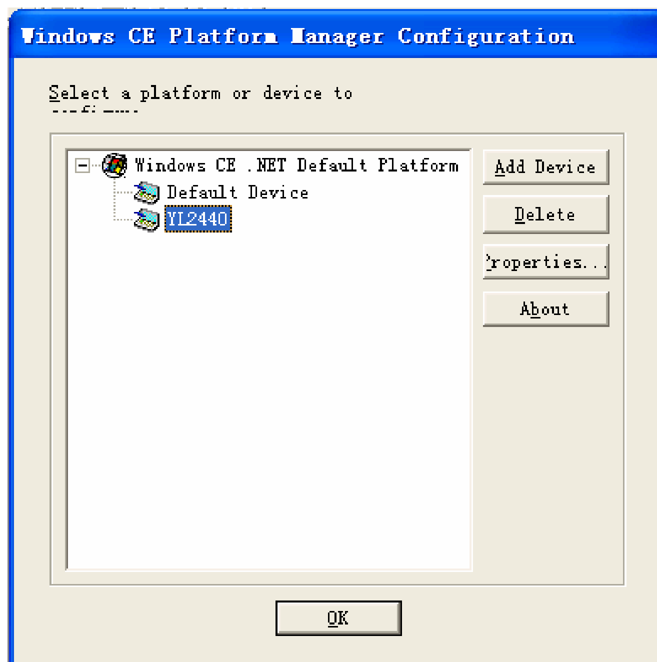
连接成功后，ActiveSync 的图标会变成另外一种颜色，并且提示连接成功。这时，打开菜单 File | Explore ，就可以浏览 WINCE 系统上的资源，也可以通过复制/粘贴的方式在系统之间拷贝文件。



ActiveSync 的成功连接是使用所有微软远程连接工具的基础。在 ActiveSync 成功连接后，就可以进一步使用到 PB 程序的“Tools”菜单项里面的许多远程工具。例如点击 PB 的 Tools ｜ Remote zoom－in 菜单，以运行远程图片缩放工具，这个程序可以实现对远程移动设备显示屏幕的截屏。Remote zoom－in 程序运行起来后，首先要配置一下平台管理器，点击它的 Connection ｜ Configure windows ce platform manager 菜单， 如下图

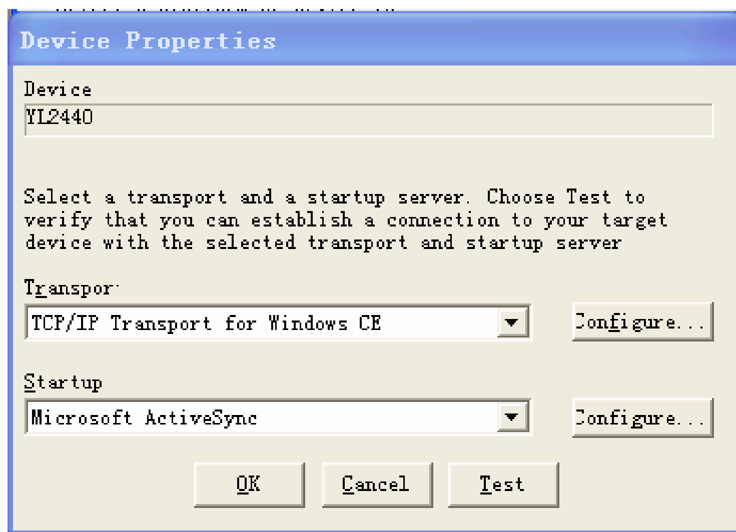


在弹出的对话框中添加一个新设备，设备名可以取为“YL2440”，如下图

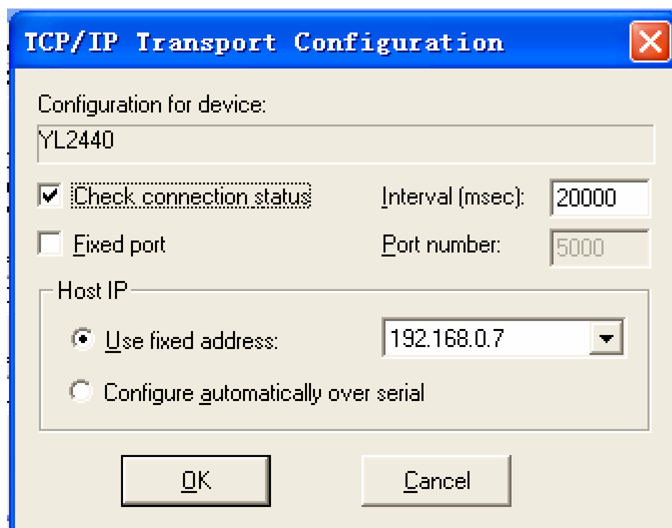


（图四十九）

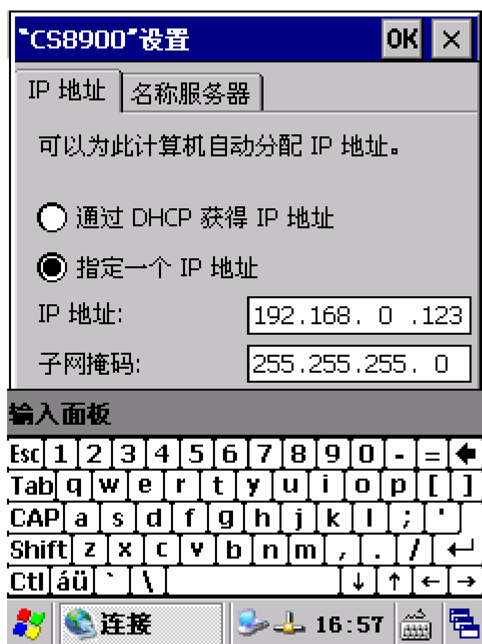
点击右边的“Propertions…”，设置“YL2440”设备的属性，如下图



点击“Transport”下拉框右边的“Configure”按钮，以设置 TCP/TP 传输的设置，HOST IP 一项填入您的桌面机器的与 WINCE 相连的以太网卡的 IP 地址，请保证 WINCE 的 IP 地址与 PC 机 IP 地址在同一段地址内，例如下图，





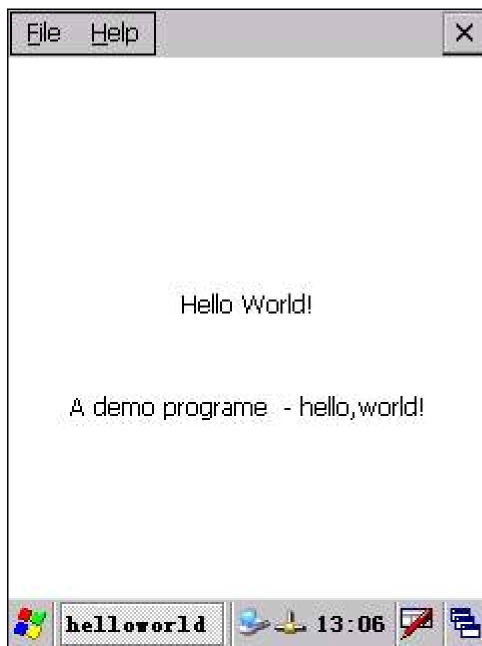


设置结束后，确信您的网线已经连好 YL2440 的网口和桌面机器的网卡，然后点击

Remote zoom—in 程序的菜单 File | New Bitmap ,开始进行截屏。



如果用 EVC 或 EVB 编译了自己编写的 WINCE 应用程序，并点击在“tools”菜单的“Configure platform manager”菜单项完成配置平台管理器后，点击运行按钮，就可以自动的把程序发送到板子上运行，此外也可以利用 sync 工具的 explorer 将编译好的程序上传到 YL2440 上运行。以下是我方编写的例子程序“Hello”在 YL2440 的 wince4.2 操作系统上执行的结果，如下图



7.5.摄像头的使用

Windows CE 启动后，在\Windows 目录下，放有摄像头测试程序 CameraTest.exe（如果没有发现，请设置显示隐藏文件；设置的方法是去掉“文件夹选项”的“不显示隐藏文件和文件夹”)。运行 CameraTest.exe，然后点击 Play 按钮.如果设备正常，这时显示捕获的图象。

附录一 YL2440 开发系统 Qt 嵌入式图形开发

（敬请注意，本公司提供的 WINCE 及 QT 开发涉及到的内容是一种友情支持，不提供电话技术支持，有需要者请参与本公司网站的论坛，敬请原谅）

基 础 篇

Qt是Trolltech公司的一个标志性产品。Trolltech公司 1994 年成立于挪威，但是公司的核心开发团队已经在 1992 年开始了Qt产品的研发，并于 1995 年推出了Qt的第一个商业版，直到现在Qt已经被世界各地的跨平台软件开发人员使用，而Qt的功能也得到了不断的完善和提高。

Qt 是一个支持多操作系统平台的应用程序开发框架，它的开发语言是C++。Qt最初主要是为跨平台的软件开发者提供统一的，精美的图形用户编程接口，但是现在它也提供了统一的网络和数据库操作的编程接口。正如微软当年为操作系统提供了友好，精致的用户界面一样，今天由于Trolltech的跨平台开发框架Qt的出现，也使得UNIX、LINUX这些操作系统以更加方便、精美的人机界面走近普通用户。

Qt是以工具开发包的形式提供给开发者的，这些工具开发包包括了图形设计器，Makefile制作工具，字体国际化工具，Qt的C++类库等等；谈到C++的类库我们自然会想到MFC，是的，Qt的类库也是等价于MFC的开发库，但是Qt的类库是支持跨平台的类库，也就是说Qt类库封装了适应不同操作系统的访问细节，这正是Qt的魅力所在。

目前，Qt可以支持的操作系统平台如下：

◆ MS/Windows 95、Windows 98、WindowsNT 4.0、Windows 2000、Windows XP；

◆ Unix/X11 Linux、Sun Solaris、HP-UX、Compaq True64Unix、IBM AIX、SGI IRIX 和很多其它X11 平台；

- Macintoshi Mac OSX；
- 嵌入式的，包含有FramBuffer的Linux平台。

Qt的资源

trolltech的主页：http://www. trolltech.com/

支持匿名访问的FTP：ftp://ftp.trolltech.com

新闻组服务器：nntp.trolltech.com

非官方的Qt文档中文翻译小组： http://www.qiliang.net/qt/index.html

1. 认识Qt/Embedded嵌入式工具开发包

1.1 介绍

Qt/Embedded 是一个为嵌入式设备上的图形用户接口和应用开发而订做的C++工具开发包. 它通常可以运行在多种不同的处理器上部署的嵌入式Linux操作系统上。如果不考虑X窗口系统的需要，居于Qt/Embedded 的应用程序可以直接对缓冲帧进行写操作。除了类库以外，Qt/Embedded还包括了几个提高开发速度的工具，使用标准的Qt API,我们可以非常熟练的在Windows和Unix编程环境里开发应用程序。

Qt/Embedded 是一组用于访问嵌入式设备的 Qt C++ API；Qt/Embedded 的Qt/X11, Qt/Windows 和Qt/Mac版本提供的都是相同的API和工具。Qt/Embedded还包括类库以及支持嵌入式开发的工具。

Qt/Embedded提供了一种类型安全的被称之为信号与插槽的真正的组件化编程机制，这种机制和以前的回调函数有所不同。Qt/Embedded还提供了一个通用的widgets类，这个类可以很容易的被子类化为客户自己的组件或是对话框。针对一些通用的任务，Qt还预先为客户定制了类似于消息框和向导这样的对话框。

运行Qt/Embedded 所需的系统资源可以很小，相对X窗口下的嵌入解决方案而言，Qt/Embedded只要求一个较小的存储空间（Flash）和内存。Qt/Embedded可以运行在不同的处理器上部署的Linux系统，只要这个系统有一个线性地址的缓冲帧并支持C++的编

译器。您可以选择不编译Qt/Embedded某些你不需要的功能，从而大大减小了它的内存占有量。

Qt/Embedded包括了它自身的窗口系统，并支持多种不同的输入设备。

开发者可以使用他们熟悉的开发环境来编写代码。Qt的图形设计器(designer)可以用来可视化地设计用户接口，设计器中有一个布局系统，它可以使您设计的窗口和组件自动根据屏幕空间的大小而改变布局。开发者可以选择一个预定义的视觉风格，或是建立自已独特的视觉风格。使用UNIX/LINUX操作系统的用户，可以在工作站上通过一个虚拟缓冲帧的应用程序仿真嵌入式系统的显示终端。

Qt/Embedded也提供了许多特定用途的非图形组件，例如国际化，网络和数据库交互组件。

Qt/Embedded是成熟可靠的工具开发包，它在世界各地被广泛使用。除了在商业上的许多应用以外，Qt/Embedded还是为小型设备提供的Qtopia应用环境的基础。Qt/Embedded以简洁的系统，可视化的表单设计和详致的API让编写代码变得愉快和舒畅。

1.2 系统要求

Qt/Embedded很省内存，因为它不需要一个X 服务器或是 Xlib库，相反它可以直接的写缓冲帧，它的内存消耗可以通过不编译某些不使用的功能来动态调节，它甚至可以把全部的应用功能编译链接到一个简单的静态链接的可执行程序中，从而能够最大的节省内存。

Qt/Embedded可以运行在被Linux支持的所有的处理器上，当然所说的“Linux支持某个处理器”是指已经有一个Linux的c＋＋编译器支持产生该处理器的目标代码以及Linux操作系统已经顺利移植到这个处理器上。目前Qt/Embedded可以运行在Intel x86,

MIPS, ARM, StrongARM, Motorola 68000 and PowerPC等处理器上。

Qt/Embedded的应用程序可以直接的写内核缓冲帧,它支持的线性的缓冲帧包括1, 4, 8, 15, 16, 24和32位深度以及VGA16的缓冲帧，任何被内核支持的图形卡也可以工作，通过对Qt/Embedded进行客户化的定制可以使得它从图形加速系统获得好处。Qt/Embedded对显示屏幕的尺寸并无限制,另外它还有许多先进的功能,例如反别名字体、alpha-blended位图和屏幕旋转等。

Qt/Embedded的库可以通过在编译时去除不需要的功能来进行精简。例如，要想不编译QlistView，可以通过定义一个QT_NO_LISTVIEW 的预处理标记来达到此目的；如果不想编译支持国际化的功能，那么可以定义QT_NO_I18N 的预处理标记。.Qt/Embedded 提供了大约200个可配置的特征，由此在Intel x86平台上库的大小范围会在700KB到5000KB之间。大部分客户选择的配置使得库的大小在1500 KB 到 4000 KB之间。

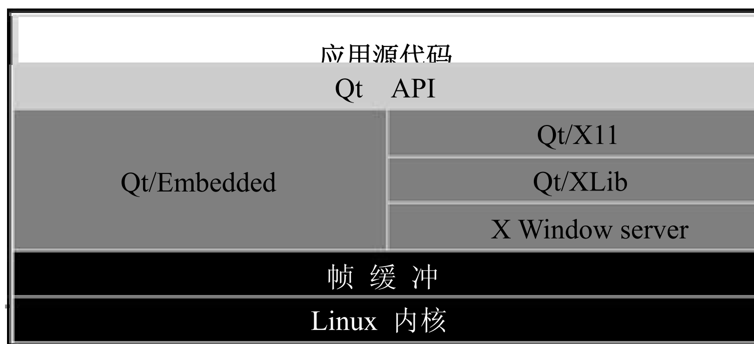
Qt/Embedded 还使用了一些节省内存空间的技术，例如隐式共享(写时复制)和缓存。在Qt中有超过20个类, 包括 QBitmap, QMap, QPalette, QPicture,QPixmap 和 QString, 使用了隐式共享技术，目的是避免不需要的复制和最小的内存需求。隐式共享过程会自动的发生，从而使编程更简单并且避免了处理指针和最优化时带来的危险。

许多的Qt组件都会被编译成为库的形式或者是插件的形式，客户视觉组件，数据库驱动，字体格式读写，图形格式转换，文本解码和窗体可以被编译成插件，从而减小核心库的大小并提供了更大的弹性。作为一种选择，如果对组件和应用了解得很深入的话，它们也可以被编译并和Qt/Embedde的库静态链接到一个简单的可执行程序，从而节省ROM,RAM和CPU的使用。

1.3 架构

Qt/Embedded 为带有轻量级窗口系统的嵌入式设备提供了一个标准的 Qt API. 。Qt/Embedded的面向对象的设计使得它一直能不断的向前支持像键盘，鼠标和图形加速卡这样的额外设备。

通过使用Qt/Embedded, 开发者可以感受到在Qt/X11, Qt/Windows和 Qt/Mac等不同的版本下使用相同的API编程带来的便利。



图一：Qt/Embedded与 Qt/X11 的Linux版本的比较

使用单一的API进行跨平台的编程可以有很多好处。提供嵌入式设备和桌面计算机环境下应用的公司可以培训开发人员使用同一套工具开发包，这有利于开发人员之间共享开发经验与知识，也使得管理人员在分配开发人员到项目中的时候增加灵活性。更远一步来说，针对某个平台而开发的应用和组件也可以销售到Qt支持的其它平台上，从而以低廉的成本扩大了产品的市场。

1.3.1 窗口系统

一个 Qt/Embedded窗口系统包含了一个或多个进程，其中的一个进程可作为服务器。该服务进程会分配客户显示区域，以及产生鼠标和键盘事件。该服务进程还能够提供输入方法和一个用户接口给运行起来的客户应用程序。该服务进程其实就是一个

有某些额外权限的客户进程。任何程序都可以在命令行上加上“－qws”的选项来把它作为一个服务器运行。

客户与服务器之间的通信使用共享内存的方法实现，通信量应该保持最小，例如客户进程直接访问帧缓冲来完成全部的绘制操作，而不会通过服务器，客户程序需要负责绘制它们自己的标题栏和其它式样。这就是Qt/Embedded库内部层次分明的处理过程。

客户可以使用QCOP通道交换消息。服务进程简单的广播QCOP消息给所有监听指定通道的应用进程，接着应用进程可以把一个插槽连接到一个负责接收的信号上，从而对消息做出响应。消息的传递通常伴随着二进制数据的传输，这是通过一个QDataStream类的序列化过程来实现的，有关这个类的的描述，见28页的“非图形类”。

QProcess类提供了另外一种异步的进程间通信的机制。它用于启动一个外部的程序并且通过写一个标准的输入和读取外部程序的标准输出和错误码来和它们通信。

1.3.2 字体

Qt/Embedded支持四种不同的字体格式：True Type字体（TTF），Postscript Type1字体，位图发布字体（BDF）和Qt 的预呈现（Pre-rendered）字体（QPF）。Qt还可以通过增加QFontFactory的子类来支持其它字体，也可以支持以插件方式出现的反别名字体。

每个TTF或者TYPE1类型的字体首次在图形或者文本方式的环境下被使用时，这些字体的字形都会以指定的大小被预先呈现出来，呈现的结果会被缓冲。根据给定的字体尺寸（例如10或12点阵）预先呈现TTF或者TYPE1类型的字体文件并把结果以QPF的格式保存，这样将可以节省内存和CPU处理时间。QPF文件包含了一些必要的字体，这些字体可以通过makeqpf工具取得，或者通过运行程序时加上“－savefonts”选项获

取。如果应用程序中使用到的字体都是QPF格式，那么Qt/Embedded将被重新配置，并排除对TTF和TYPE1类型的字体的编译，这样就可以减少Qt/Embedded的库的大小和存储字体的空间。例如一个10点阵大小的包含所有ASII字符的QPF字体文件的大小为1300字节，这个文件可以直接从物理存储格式映射成为内存存储格式。

Qt/Embedded的字体通常包括Unicode字体的一部分子集，ASII和Latin-1。一个完整的16点阵的Unicode字体的存储空间通常超过1M，我们应尽可能存储一个字体的子集，而不是存储所有的字，例如在您的应用中，您仅仅需要以Cappuccino字体、粗体的方式显示您的产品的名称，但是您却有一个包含了全部字形的字体文件。

1.3.3 输入设备

Qt/Embedded 3.0支持几种鼠标协议：BusMouse, IntelliMouse,Microsoft和MouseMan.Qt/Embedded 还支持 NEC Vr41XX 和 iPAQ 的触摸屏。通过从QWSMouseHandler或者QcalibratedMouseHandler派生子类，开发人员可以让Qt/Embedded支持更多的客户指示设备。

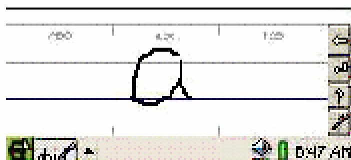
Qt/Embedded支持标准的101键盘和Vr41XX按键，通过子类化QWSKeyboardHandler 可以让Qt/Embedded支持更多的客户键盘和其它的非指示设备。

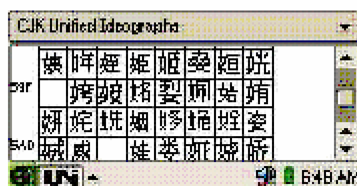
1.3.4 输入方法

对于非拉丁语系字符（例如阿拉伯，中文，希伯来和日语）的输入法，需要把它写成过滤器的方式，并改变键盘的输入。输入法的作者应该对全部的Qt API的使用有完全的认识。

在一个无键盘的设备上，输入法成了唯一的输入字符的手段。Qtpia提供了四种输入方法：笔迹识别器，图形化的标准键盘，Unicode键盘，居于字典方式提取的键盘。这些

键盘的样式如下所示

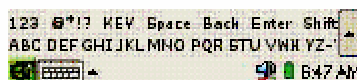

笔迹识别器



Unicode 输入



标准键盘



字典式提取键盘

1.3.5 屏幕加速

通过子类化QScreen 和 QgfxRaster 可以实现硬件加速，从而为屏幕操作带来好处。Trolltech提供了Mach64 和 Voodoo3 视频卡的硬件加速的驱动例子，同时可以按照协议编写其它的驱动程序。

入 门 篇

1. Qt/Embedded 开发环境的安装

一般来说，居于Qt/Embedded开发的应用程序最终会发布到安装有嵌入式Linux操作系统的小型设备上，所以使用装有Linux操作系统的PC机或者工作站来完成Qt/Embedded开发当然是最理想的环境，尽管Qt/Embedded也可以安装在Unix和

Windows系统上。

下面将介绍如何在一台装有Linux操作系统的机器上建立Qt/Embedded开发环境。

首先，您需要拥有三个软件安装包：tmake工具安装包，Qt/Embedded 安装包，Qt的X11 版的安装包。由于上述这些软件安装包有许多不同的版本，您要注意由于版本的不同导致这些软件在使用时可能造成的冲突，为此将告诉您一些基本的安装原则：当您选择或下载了Qt/Embedded 的某个版本的安装包之后，您下一步要选择安装的Qt for X11 的安装包的版本必须比您最先下载的Qt/Embedded 的版本要旧，这是因为Qt for X11 的安装包的两个工具uic和designer产生的源文件会和Qt/Embedded的库一起被编译链接，本着“向前兼容”的原则，Qt for X11 的版本应比Qt/Embedded的版本旧。

将以下面所列版本的安装包，一步一步介绍Qt/Embedded开发环境建立的过程（这些软件可以免费从trolltech的WEB或FTP服务器上下载），

- tmake 1.11 或更高版本；　（生成Qt/Embedded应用工程的Makefile文件）
- Qt/Embedded 2.3.7 （Qt/Embedded 安装包）
- Qt 2.3.2 for X11；　（Qt的X11 版的安装包，它将产生x11 开发环境所需要的两个工具）

1.1 安装tmake

在Linux命令模式下运行以下命令：

```
tar xfz tmake-1.11.tar.gz
export TMAKEDIR=$PWD/tmake-1.11
export TMAKEPATH=$TMAKEDIR/lib/qws/linux-x86-g++
export PATH=$TMAKEDIR/bin:$PATH
```

1.2 安装Qt/Embedded 2.3.7

在Linux命令模式下运行以下命令：

```
tar xfz qt-embedded-2.3.7.tar.gz
cd qt-2.3.7
EXPORT QTDIR=$PWD
export QTEDIR=$QTDIR
export PATH=$QTDIR/bin:$PATH
export LD_LIBRARY_PATH=$QTDIR/lib:$LD_LIBRARY_PATH
./configure －qconfig local－qvfb -depths 4,8,16,32
make sub-src
cd ..
```

上述命令 ./configure -qconfig -qvfb -depths 4,8,16,32 指定Qt嵌入式开发包生成虚拟缓冲帧工具qvfb，并支持 4，8，16，32 位的显示颜色深度。另外我们也可以在configure的参数中添加－system－jpeg和gif，使Qt/Embedded平台能支持jpeg、gif格式的图形。

上述命令 make sub-src 指定按精简方式编译开发包，也就是说有些Qt类未被编译。Qt嵌入式开发包有 5 种编译范围的选项，使用这些选项，可控制Qt生成的库文件的大小，但是您的应用所使用到的一些Qt类将可能因此在Qt的库中找不到链接。编译选项的具体用法可运行./configure －help命令查看。

1.3 安装Qt/X11 2.3.2

在Linux命令模式下运行以下命令：

```
tar xfz qt-x11-2.3.2.tar.gz
cd qt-2.3.2
```

```
export QTDIR=$PWD
export PATH=$QTDIR/bin:$PATH
export LD_LIBRARY_PATH=$QTDIR/lib:$LD_LIBRARY_PATH
./configure -no-opengl
make
MAKE -C TOOLS/QVFB
mv tools/qvfb/qvfb bin
cp bin/uic $QTEDIR/bin
cd ..
```

根据开发者本身的开发环境，也可以在configure的参数中添加别的参数，比如-no-opengl或-no-xfs，可以键入./configure -help来获得一些帮助信息。

2.认识Qt/Embedded 开发环境

Qt/Embedded的开发环境可以取代那些我们熟知的UNIX和WINDOWS开发工具。它提供了几个跨平台的工具使得开发变得迅速和方便，尤其是它的图形设计器。Unix下的开发者可以在PC机或者工作站使用虚拟缓冲帧，从而可以仿真一个和嵌入式设备的显示终端大小，象素相同的显示环境。

嵌入式设备的应用可以在安装了一个跨平台开发工具链的不同的平台上编译。最通常的做法是在一个UNIX系统上安装跨平台的带有libc库的GNU c++编译器和二进制工具。在开发的许多阶段，一个可替代的做法是使用Qt的桌面版本，例如Qt/X11 或是Qt/Windows来进行开发。这样开发人员就可以使用他们熟悉的开发环境，例如微软的Visual C++ 或者 Borland C++；在UNIX操作系统下，许多环境也是可用的，例如Kdevelop，它也支持交互式开发。

如果Qt/Embedded的应用是在UNIX平台下开发的话，那么它就可以在开发的机器上以一个独立的控制台或者虚拟缓冲帧的方式来运行，对于后者来说，其实是有一个

X11 的应用程序虚拟了一个缓冲帧。通过指定显示设备的宽度，高度和颜色深度，虚拟出来的缓冲帧将和物理的显示设备在每个像素上保持一致。这样每次调试应用时开发人员就不用总是刷新嵌入式设备的FLASH存储空间，从而加速了应用的编译、链接和运行周期。

运行Qt的虚拟缓冲帧工具的方法是：在Linux的图形模式下运行命令：

```
qvfb (回车)
```

当Qt嵌入式的应用程序要把显示结果输出到虚拟缓冲帧时，我们在命令行运行这个程序时，在程序名后加上-qws的选项。例如： $> hello －qws

2.1 QT的支撑工具

Qt包含了许多支持嵌入式系统开发的工具，其中一些工具我们会在别的地方介绍。有两个最实用的工具（除了上面我们提到的虚拟缓冲帧）是qmake和Qt designer(图形设计器)。

qmake是一个为编译Qt/Embedded库和应用而提供的Makefile生成器。它能够根据一个工程文件（.pro）产生不同平台下的Makefile文件。qmake支持跨平台开发和影子生成(shadow builds)，影子生成是指当工程的源代码共享给网络上的多台机器时，每台机器编译链接这个工程的代码将在不同的子路径下完成，这样就不会覆盖别人的编译链接生成的文件。qmake还易于在不同的配置之间切换。

开发者可以使用Qt图形设计器可视化地设计对话框而不需编写一行代码。使用Qt图形设计器的布局管理可以生成具有平滑改变尺寸的对话框，qmake和Qt图形设计器

是完全集成在一起的。

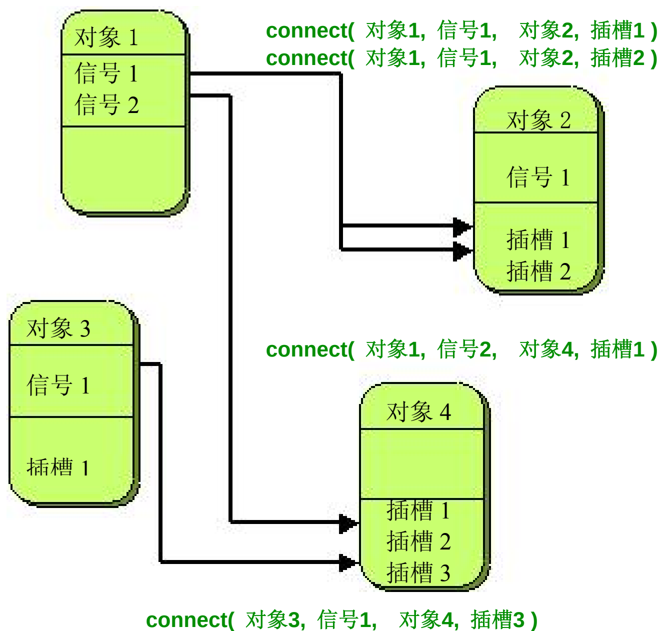
2.2 信号与插槽

信号与插槽机制提供了对象间的通信机制，它易于理解和使用，并完全被Qt图形设计器所支持。

图形用户接口的应用需要对用户的动作做出响应。例如，当用户点击了一个菜单项或是工具栏的按钮时，应用程序会执行某些代码。大部分情况下，是希望不同类型的对象之间能够进行通信。程序员必须把事件和相关代码联系起来，这样才能对事件做出响应。以前的工具开发包使用的事件响应机制是易崩溃的，不够健壮的，同时也不是面向对象的。Trolltech已经创立了一种新的机制，叫做“信号与插槽”。信号与插槽是一种强有力的对象间通信机制，它完全可以取代原始的回调和消息映射机制；信号与插槽是迅速的，类型安全的，健壮的，完全面向对象并用C++来实现的一种机制。

在以前，当使用回调函数机制来把某段响应代码和一个按钮的动作相关联时，通常把那段响应代码写成一个函数，然后把这个函数的地址指针传给按钮，当那个按钮被按下时，这个函数就会被执行。对于这种方式，以前的开发包不能够确保回调函数被执行时所传递进来的函数参数就是正确的类型，因此容易造成进程崩溃，另外一个问题是，回调这种方式紧紧的绑定了图形用户接口的功能元素，因而很难把开发进行独立的分类。

Qt的信号与插槽机制是不同的。Qt的窗口在事件发生后会激发信号。例如一个按钮被点击时会激发一个“clicked”信号。程序员通过建立一个函数（称作一个插槽），



图一 一些信号与插槽连接的抽象图

然后调用connect()函数把这个插槽和一个信号连接起来，这样就完成了一个事件和响应代码的连接。信号与插槽机制并不要求类之间互相知道细节，这样就可以相对容易的开发出代码可高重用的类。信号与插槽机制是类型安全的，它以警告的方式报告类型错误，而不会使系统产生崩溃。

例如，如果一个退出按钮的clicked() 信号被连接到了一个应用的退出函数－quit() 插槽。那么一个用户点击退出键将使应用程序终止运行。上述的连接过程用代码写出来就是这样

```
connect( button, SIGNAL(clicked()), qApp, SLOT(quit()) );
```

可以在Qt应用程序的执行过程中增加或是减少信号与插槽的连接。

信号与插槽的实现扩展了C++的语法，同时也完全利用了C++面向对象的特征。信号与插槽可以被重载或者重新实现，它们可以定义为类的公有，私有或是保护成员。

2.2.1 信号与插槽的例子

如果一个类要使用信号与插槽机制，它就必须是从QObject或者QObject的子类继承，而且在类的定义中必须加上Q_OBJECT宏。信号被定义在类的信号部分，而插槽则定义在public slots, protected slots 或者 private slots 部分。

下面定义一个使用到信号与插槽机制的类。

```
class BankAccount : public QObject
{
    Q_OBJECT
    public:
        BankAccount() { curBalance = 0; }
        int balance() const { return curBalance; }
    public slots:
        void setBalance( int newBalance );
    signals:
        void balanceChanged( int newBalance );
    private:
        int curBalance;
};
```

和大部分的 C++的类一样，BankAccount 类有一个构造函数，还有一个取值的函数 balance()，一个设置值的函数 setBalance(int newBalance)。

这个类有一个信号balanceChanged(),这个信号声明了它在BankAccount类的成员curBalance的值被改变时产生。信号不需要被实现，当信号被激发时，和该信号连接的插槽将被执行。

上面用来设置值的函数 setBalance(int newBalance)定义在类的“public slots”部分，因此它是一个插槽。插槽是一个需要实现的标准的成员函数，它可以像其它函数一样被调用，也可以和信号相连接。

下面就是该插槽函数 setBalance(int newBalance)的实现代码：

```
void BankAccount::setBalance( int newBalance )
{
    if ( newBalance != curBalance )
    {
        curBalance = newBalance;
        emit balanceChanged( curBalance );
    }
}
```

其中的一段代码

emit balanceChanged(curBalance);

它的作用是当 curBalance 的值被改变后，将新的 curBalance 的值作为参数去激活 balanceChanged（）信号。对于关键词“emit”，它和信号、插槽一样是由 Qt 提供的，这些关键词都会被 c＋＋的预处理机制转换为 c＋＋代码。

一个对象的信号可以被多个不同的插槽连接，而多个信号也可以被连接到相同的插槽。当信号和插槽被连接起来时，应当确保它们的参数类型是相同的，如果插槽的参数个数小于和它连接在一起的信号的参数个数，那么从信号传递插槽的多余的参数将被忽略。

2.2.2 元对象编译器

信号与插槽机制是以纯 C++代码来实现的，实现的过程使用到了 Qt 开发工具包提供的预处理器和元对象编译器（moc）。

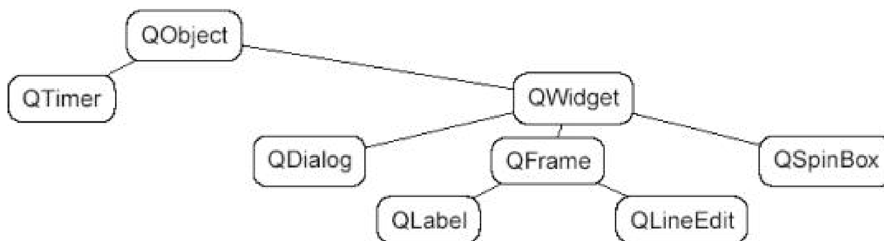
moc 读取应用程序的头文件，并产生支持信号与插槽的必要的代码。开发者没必要编辑或是浏览这些自动产生的代码，当有需要时，qmake 生成的 Makefile 文件里会显式的包含了运行 moc 的规则。

除了可以处理信号与插槽机制之外，moc 还支持翻译机制，属性系统和运行时的信息。

2.3 窗体

Qt 拥有丰富的满足不同需求的窗体（按钮，滚动条等等），Qt 的窗体使用起来很灵活，为了满足特别的要求，它很容易就可以被子类化。

窗体是**Qwidget**类或它子类的实例，客户自己的窗体类需要从**Qwidget**它的子类继承。



图二 摘录的 Qwidget 类的继承图

一个窗体可以包含任意数量的子窗体，子窗体可以显示在父窗体的客户区，一个没父窗体的窗体我们称之为顶级窗体（一个“窗口”），一个窗体通常有一个边框和标题栏作为装饰。Qt 并未对一个窗体有什么限制，任何类型的窗体可以是顶级窗体，任何类型的窗体可以是别的窗体的子窗体。在父窗体显示区域的子窗体的位置可以通过布局管理自动的进行设置，也可以人为的指定。当父窗体无效，隐藏或被删除后，它的子窗体都会进行同样的动作。

标签，消息框，工具栏等等等，并未被限制使用什么颜色，字体和语言。Qt的文本呈现窗体可以使用HTML子集显示一个多语言的宽文本。

2.3.1 一个 Hello 的例子

下面是一个显示“Hello Qt/Embedded!”的程序的完整的源代码：



图三 Hello Qt/Embedded

```
#include <qapplication.h>
#include <qlabel.h>
int main( int argc, char **argv )
{
    QApplication app( argc, argv );
    QLabel *hello = new QLabel( "<font color=blue>Hello"
    " <i>Qt/Embedded!</i></font>", 0 );
    app.setMainWidget( hello );
    hello->show();
    return app.exec();
}
```

2.3.2 通用窗体

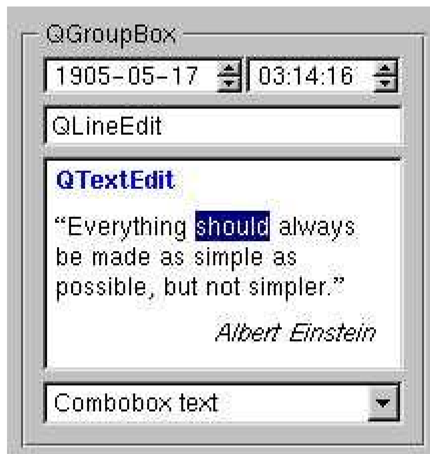
下面是一些主要的 Qt 窗体的截屏图，这些窗体使用了窗口样式。



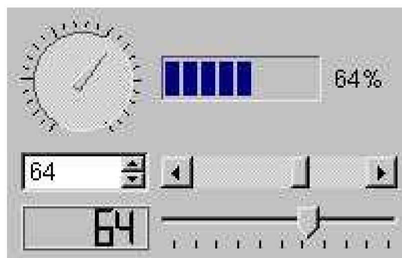
图四 使用了 QHBox 进行排列一个标签和一个按钮



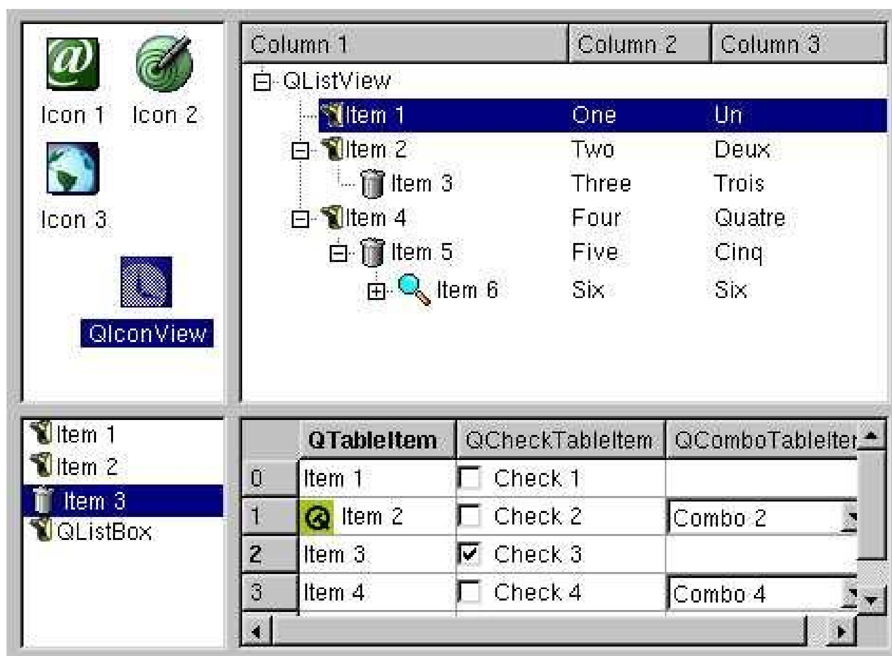
图五 使用了QbuttonGroup的两个单选框和两个复选框



图六 使用了QgroupBox进行排列的的日期类QDateTimeEdit, 一个行编辑框类QLineEdit, 一个文本编辑类QTextEdit 和一个组合框类QComboBox



图七 以QGrid排列的一个 QDial, 一个QProgressBar, 一个QSpinBox, 一个QScrollBar, 一个QLCDNumber和一个Qslider



图八 以QGrid排列的一个QIconView, 一个 QListView, 一个 QListBox 和一个 QTable

有些时候在进行字符输入时，希望输入的字符满足了某种规则才能使输入被确认。Qt提供了解决的办法，例如QComboBox, QLineEdit 和 QspinBox 的字符输入可以通过

Qvalidator的子类来进行约束和有效性检查。

通过继承QScrollView ，QTable, QListView, QTextEdit 和其它窗体就能够显示大量的数据，并且自动的拥有了一个滚动条。

许多Qt创建的窗体能够显示图像，例如按钮，标签，菜单项等等。Qimage类支持几种图形格式的输入、输出和操作，它目前支持的图形格式有BMP, GIF*, JPEG, MNG, PNG, PNM, XBM 和 XPM。

2.3.3 画布

QCanvas 类提供了一个高级的平面图形编程接口，它可以处理大量的像线条、矩形、椭圆、文本、位图、动画等这些画布项，画布项可以较容易的做成交互式的（例如做成支持用户移动的）。

画布项是QcanvasItem子类的实例，它们比窗体类Qwidget更显得轻量级，它们能够被快速的移动，隐藏和显示。Qcanvas可以更有效的支持冲突检测，它能够列出一个指定区域里面的所有的画布项。QcanvasItem可以被子类化，从而可以提供更多的客户画布项类型，或者扩展已有的画布项的功能。

Qcanvas对象是由QcanvasView进行绘制的，QcanvasView对象可以以不同的译文、比例、旋转角度，剪切方式去显示同一个画布。

Qcanvas 对象是理想的数据表现方式，它已经被消费者用于绘制地图和显示网络拓扑结构。它也可用于制作快节奏的且有大量角色的平面游戏。



图九 在 Qtopia 中用 QCanvas 实现的小行星游戏

2.3.4 客户窗体

通过对Qwidget或者它的子类进行子类化，可以建立自己的客户窗体或者对话框。下面是一个完整的源代码例子，它示例了如何通过子类化窗体，绘制一个模拟的时钟。AnalogClock 窗体类是Qwidget的子类，它显示当前时间，并且可以自动地更新时间。



图十 模拟钟窗体

在 analogclock.h头文件中，**AnalogClock** 以这样地形式定义：:

```
#include <qwidget.h>
class AnalogClock : public QWidget
{
    public:
    AnalogClock( QWidget *parent = 0, const char *name = 0 );
    protected:
    virtual void timerEvent( QTimerEvent *event );
    virtual void paintEvent( QPaintEvent *event );
};
```

AnalogClock 类继承了Qwidget，它有一个典型的窗体类构造函数，这个函数有父窗口对象指针和名字指针两个参数。(如果设置了名字的话，测试和调试起来就会容易些) timerEvent() 函数是从QObject （Qwidget的父类）对象继承而来的，这个函数会被系统定期调用。paintEvent() 函数是从QWidget 继承而来的并且当窗体需要重画时这个函数就会被调用。

timerEvent() 和 paintEvent() 函数是“事件句柄”的两个例子。应用对象以重载父类对象的虚拟函数events (QEvent objects) 的形式接收系统的事件。大约有超过50个的系统事件是较常用的，例如 MouseButtonPress, MouseButtonRelease, KeyPress, KeyRelease, Paint, Resize 和Close. 对象可以对发给它们的事件做出响应或者筛选一些事件后再发送给别的对象。

analogclock.cpp 文件是定义在analogclock.h中的函数的实现源文件

```
#include <qdatetime.h>
#include <qpainter.h>
#include "analogclock.h"
```

```
AnalogClock::AnalogClock( QWidget *parent, const char *name )
: QWidget( parent, name )
{
    startTimer( 12000 );
    resize( 100, 100 );
}
void AnalogClock::timerEvent( QTimerEvent * )
{
    update();
}
void AnalogClock::paintEvent( QPaintEvent * )
{
    QCOORD hourHand[8] = { 2, 0, 0, 2, -2, 0, 0, -25 };
    QCOORD minuteHand[8] = { 1, 0, 0, 1, -1, 0, 0, -40 };
    QTime time = QTime::currentTime();
    QPainter painter( this );
    painter.setWindow( -50, -50, 100, 100 );
    painter.setBrush( black );
    for ( int i = 0; i < 12; i++ )
    {
        painter.drawLine( 44, 0, 46, 0 );
        painter.rotate( 30 );
    }
    painter.save();
    painter.rotate( 30 * (time.hour() % 12) + time.minute() / 2 );
    painter.drawConvexPolygon( QPointArray(4, hourHand) );
    painter.restore();
    painter.save();
    painter.rotate( 6 * time.minute() );
    painter.drawConvexPolygon( QPointArray(4, minuteHand) );
    painter.restore();
}
```

构造函数设置窗口的尺寸大小为100 x 100，并且告诉系统每隔12秒调用一次timerEvent()函数，从而对模拟钟的窗体进行刷新。

在 timerEvent()函数中， 通过调用QWidget的函数 update()就可以告诉Qt，窗体需要立即重画，紧接着Qt就会产生一个绘制事件并且调用paintEvent()函数。

在paintEvent()函数中，一个Qpainter对象用于在窗体上绘制12个刻度以及分针，时针。Qpainter类提供了一种统一的方式用于绘制窗体，位图，矢量图等，它提供了绘制点，线，椭圆，多边形，弧，贝塞尔曲线等功能，一个Qpainter的坐标系可以被转变，缩放，旋转，和剪切，这样对象就可以根据它在窗口或者窗体上的位置绘制出一个剪切的视图。剪切可以使窗体绘制时减少闪烁。使用QPainter 的子类QdirectPainter可以锁定和直接访问帧缓冲区域。

文件 analogclock.h 和 analogclock.cpp 完全的定义和实现了AnalogClock 客户窗体类，这个窗体是现在就可以使用的。

```
#include <qapplication.h>
#include "analogclock.h"
int main( int argc, char **argv )
{
    QApplication app( argc, argv );
    AnalogClock *clock = new AnalogClock;
    app.setMainWidget( clock );
    clock->show();
    return app.exec();
}
```

2.3.5 主窗口

QMainWindow类是为应用的主窗口提供一个摆放相关窗体的框架。

一个主窗口包含了一组标准窗体的集合。主窗口的顶部包含一个菜单栏，它的下方放置着一个工具栏，工具栏可以移动到其它的停靠区域。主窗口允许停靠的位置有顶部，左边，右边和底部。工具栏可以被拖放到一个停靠的位置，从而形成一个浮动的工具面板。主窗口的下方，也就是在底部的停靠位置之下有一个状态栏。主窗口的中间区域可以包含其它的窗体。提示工具和“这是什么”帮助按钮以旁述的方式阐述了用户接口的使用方法。

对于小屏幕的设备，使用Qt图形设计器定义的标准的Qwidget模板比使用主窗口类更好一些。典型的模板包含有菜单栏，工具栏，可能没有状态栏（在必要的情况下，可以用任务栏，标题栏来显示状态）

2.3.6 菜单

弹出式菜单QpopupMenu类以垂直列表的方式显示菜单项，它可以是单个的（例如上下文相关菜单），可以以菜单栏的方式出现，或者是别的弹出式菜单的子菜单出现。

每个菜单项可以有一个图标，一个复选框和一个加速器（快捷键），菜单项通常对应一个动作（例如存盘），分隔器通常显示成一条竖线，它用于把一组相关联的动作菜单分立成组。

下面是一个建立包含有 New，Open 和 Exit 菜单项的文件菜单的例子。

```
QPopupMenu *fileMenu = new QPopupMenu( this );
fileMenu->insertItem( "&New", this, SLOT(newFile()), CTRL+Key_N );
fileMenu->insertItem( "&Open...", this, SLOT(open()), CTRL+Key_O );
fileMenu->insertSeparator();
fileMenu->insertItem( "E&xit", qApp, SLOT(quit()), CTRL+Key_Q );
```

当一个菜单项被选中，和它相关的插槽将被执行。加速器(快捷键)很少在一个没有键盘

输入的设备上使用，Qt/Embedded 的典型配置并未包含对加速器的支持。上面出现的代码“&New”意思是在桌面机器上以“New”的方式显示出来，但是在嵌入式设备上，它只会显示为“New”。

QmenuBar类实现了一个菜单栏，它会自动的设置几何尺寸并在它的父窗体的顶部显示出来，如果父窗体的宽度不够宽以致不能显示一个完整的菜单栏，那么菜单栏将会分为多行显示出来。Qt内置的布局管理能够自动的调整菜单栏。

Qt的菜单系统是非常灵活的，菜单项可以被动态的使能，失效，添加或者删除。通过子类化QcustomMenuItem，我们可以建立客户化外观和功能的菜单项。

2.3.7 工具栏

QtoolButton类实现了一个带有图标，3维边框和可选标签的工具栏按钮。切换工具栏按钮具有开、关的特征，其它的按钮则执行一个命令。不同的图标用来表示按钮的活动，无效、使能模式，或者是开或关的状态。如果你仅为按钮指定了一个图标，那么Qt会使用可视提示来表现按钮不同的状态，例如按钮失效时显示灰色。

工具栏按钮通常以一排的形式显示在工具栏上，对于一个有几组工具栏的应用，用户可以随便的到处移动这些工具栏，工具栏差不多可以包含所有的窗体，例如QComboBoxes 和 QspinBoxes。

2.3.8 旁述

现代的应用主要使用旁述的方式去解释用户接口的用法。Qt 提供了两种旁述的方式：“提示栏”和“这是什么”帮助按钮。

“提示栏”是小的，通常是黄色的矩形，当鼠标在窗体的一些位置游动时它就会

自动出现。它主要用于解释工具栏按钮，特别是那些缺少文字标签说明的工具栏按钮的用途。下面就是如何设置一个“存盘”按钮的提示的代码。

```
QToolTip::add( saveButton, ″Save″ );
```

当提示字符出现之后，你还可以在状态栏显示更详细的文字说明。

对于一些没有鼠标的设备（例如那些使用触点输入的设备），就不会有鼠标的光标在窗体上进行游动，这样就不能激活提示栏。对于这些设备也许就需要使用“这是什么”帮助按钮，或者使用一种姿态来表示输入设备正在进行游动，例如用按下或者握住的姿态来表示现在正在进行游动。

“这是什么”帮助按钮和提示栏有些相似，只不过前者是要用户点击它才会显示旁述。在小屏幕设备上，要想点击“这是什么”帮助按钮，具体的方法是，在靠近应用的 X 窗口的关闭按钮“x”附近你会看到一个“？”符号的小按钮，这个按钮就是“这是什么”帮助按钮。一般来说，“这是什么”帮助按钮按下后要显示的提示信息应该比提示栏要多一些。下面是设置一个存盘按钮的“这是什么”文本提示信息的方法:

```
QWhatsThis::add( saveButton, ″Saves the current file.″ );
```

QToolTip 和 QWhatsThis 类提供了虚拟函数以供开发者重新实现更多的特定的用途。

Qtopia并未使用上述提及的两种帮助（旁述）机制。它在应用窗口的标题栏上放置一个“？”符号的按钮来代替上述的旁述机制，这个“？”按钮可以启动一个浏览器来显示和当前应用相关的HTML页面。Qtopia使用按下和握住的姿态来调用上下文菜单（右击）和属性对话框。

2.3.9 动作

应用程序通常提供给用户几种不同的方式去执行特别的动作。例如，大部分应用提供了一个“Save”动作给用于存盘的菜单(File|Save)以及工具栏（一个“软盘”图标的工具栏按钮）和快捷键(Ctrl+S)。Qaction类可以让上述过程变得简洁，它允许程序员在一个地方定义一个动作，然后把这个动作加入到菜单或者工具栏，这个过程与把菜单项加入到菜单的道理是一样的。

下面的代码实现了一个“Save”菜单项和一个“Save”工具按钮，旁述系统和快捷键可以很容易的添加进去，但是我们没添加，因为它们很少在嵌入式设备上使用。

```
QAction *saveAct = new QAction( this );
saveAct->setText( "Save" );
saveAct->setIconSet( QPixmap("save.png") );
connect( saveAct, SIGNAL(activated()), this, SLOT(save()) );
saveAct->addTo( fileMenu );
saveAct->addTo( toolbar );
```

除了不用复制代码，使用 Qaction 还可以确保菜单的状态与工具栏按钮的状态保持一致，必要的时候还可显示提示栏。使一个动作（Action）失效将导致和该动作相关联的菜单项以及工具按钮的失效。同样的，如果用户切换一个工具按钮的状态，那么相关的菜单项的也会跟着被选中或不选中。

2.4 对话框

使用 Qt 图形设计器这个可视化设计工具用户可以建立自己的对话框。Qt 使用布局管理自动的设置窗体与别的窗体之间相对的尺寸和位置，这样可以确保对话框能够

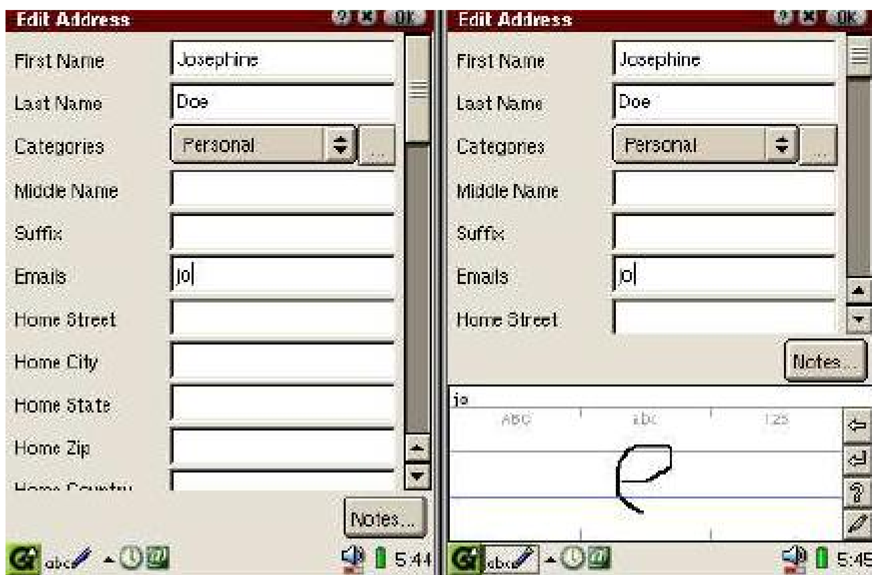
最好的利用屏幕上的可用空间。使用布局管理意味着按钮和标签可以根据要显示的文字自动的改变自身大小，而用户完全不用考虑文字是那一种语言。

2.4.1 布局

Qt 的布局管理用于组织管理一个父窗体区域内的子窗体。它的特点是可以自动的设置子窗体的位置和大小，并可判断出一个顶级窗体的最小和缺省的尺寸，当窗体的字体或内容变化后，它可以重置一个窗体的布局。

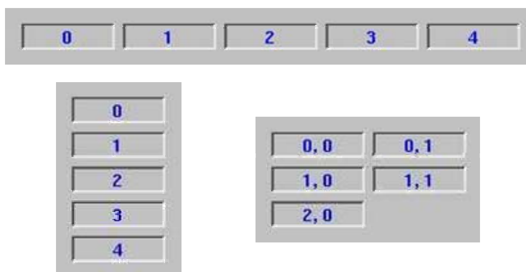
使用布局管理，开发者可以编写独立于屏幕大小和方向之外的程序，从而不需要浪费代码空间和重复编写代码。对于一些国际化的应用程序，使用布局管理，可以确保按钮和标签在不同的语言环境下有足够的空间显示文本，不会造成部分文字被剪掉。

布局管理使得提供部分用户接口组件，例如输入法和任务栏变得更容易。我们可以通过一个例子说明这一点，当Qtopia的用户正在输入文字时，输入法会占用一定的文字空间，应用程序这时也会根据可用的屏幕尺寸的变化调整自己。



图十 Qtopia 的布局管理

Qt提供了三种用于布局管理的类：QHBoxLayout，QVBoxLayout 和 QgridLayout。



图十一 QHBoxLayout，QVBoxLayout 和 QGridLayout 的布局效果

QHBoxLayout 布局管理把窗体按照水平方向从左至右排成一行

QVBoxLayout 布局管理把窗体按照垂直方向从上至下排成一列

QGridLayout 布局管理以网格的方式来排列窗体，一个窗体可以占据多个网格。

在多数情况下，布局管理在管理窗体时执行最优化的尺寸，这样窗口看起来就更

好看而且可以尺寸变化会更平滑。使用以下的机制可以简化窗口布局的过程：

1、为一些子窗口设置一个最小的尺寸，一个最大的或者固定的尺寸。

2、增加拉伸项（stretch items）或者间隔项（spacer item）。拉伸项和间隔项可以填充一个排列的空间。

3、改变子窗口的尺寸策略，程序员可以调整窗体尺寸改变时的一些策略。子窗体可以被设置为扩展，紧缩和保持相同尺寸等策略。

4、*改变子窗口的尺寸提示。QWidget::sizeHint() 和 QWidget::minimumSize-Hint() 函数返回一个窗体根据自身内容计算出的首选尺寸和首选最小尺寸，我们在建立窗体时可考虑重新实现这两个函数。*

5、设置拉伸比例系数。设置拉伸比例系数是指允许开发者设置窗体之间占据空间大小的比例系数，例如我们设定可用空间的 2/3 分配给窗体 A，剩下的 1/3 则分配给窗体 B。

布局管理也可按照从右至左，从下到上的方式来进行。当一些国际化的应用需要支持从右至左阅读习惯的语言文字（例如阿拉伯和希伯来）时，使用从右至左的布局排列是更方便的。

布局是可以嵌套的和随意进行的。下面是一个对话框的例子，它以两种不同尺寸大小来显示：



图十二 小的对话框和大的对话框

这个对话框使用了三种排列方式。QVBoxLayout管理一组按钮，QHBoxLayout管理一个显示国家名称的列表框和右边那组按钮，QVBoxLayout管理窗体上剩下的组件“Now please select a country”标签。在“< Prev”和“Help”按钮之间放置了一个拉伸项（stretch items），使得两者之间保持了一定比例的间隔。

建立这个对话框窗体和布局管理的实现代码如下：

```
QVBoxLayout *buttonBox = new QVBoxLayout( 6 );
buttonBox->addWidget( new QPushButton("Next >", this) );
buttonBox->addWidget( new QPushButton("< Prev", this) );
buttonBox->addStretch( 1 );
buttonBox->addWidget( new QPushButton("Help", this) );
QListBox *countryList = new QListBox( this );
countryList->insertItem( "Canada" );
/* … */
```

```
countryList->insertItem( "United States of America" );
QHBoxLayout *middleBox = new QHBoxLayout( 11 );
middleBox->addWidget( countryList );
middleBox->addLayout( buttonBox );
QVBoxLayout *topLevelBox = new QVBoxLayout( this, 6, 11 );
topLevelBox->addWidget( new QLabel("Now please select a country", this) );
topLevelBox->addLayout( middleBox );
```

使用 Qt 图形设计器设计的这个对话框，显示如下



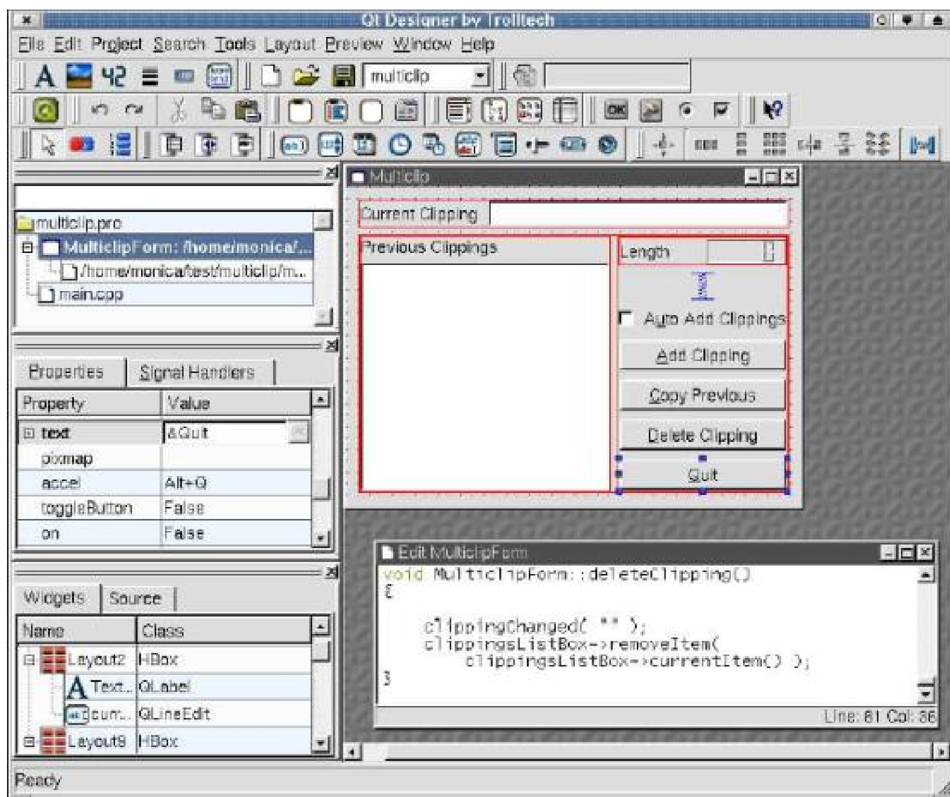
图十三 Qt 图形设计器中使用了布局的对话框

2.4.2 Qt 图形设计器

Qt 图形设计器是一个具有可视化用户接口的设计工具。Qt 的应用程序可以完全用源代码来编写，或者使用 Qt 图形设计器来加速开发工作。启动 Qt 图形设计器的方法是：在 Linux 命令模式下，键入以下命令(假设 Qt X11 安装在/usr/local 下)：

```
cd qt-2.3.2/bin
  ./designer
```

这样就可以启动一个与 Windows 下的 Delphi 相类似界面。 下图是使用 Qt 图形设计器设计一个表单的截屏图。



图十四 Qt 图形设计器

开发者点击工具栏上的代表不同功能的子窗体/组件的按钮，然后把它放到一个

表单上面，这样就可以把一个子窗体/组件放到表单上了。开发者可以使用属性对话框来设置子窗体的属性。精确的设置子窗体的位置和尺寸大小是没必要的。开发者可以选择一组窗体，然后对他们进行排列。例如，我们选定了一些按钮窗体，然后使用“水平排列（lay out horizontally）”选项对它们进行一个接一个的水平排列。这样做使得设计工作变得更快，而且完成后的窗体将能够按照属性设置的比例填充窗口的可用尺寸范围。

使用 Qt 图形设计器进行图形用户接口的设计可以消除应用的编译，链接和运行时间，同时使得修改图形用户接口的设计变得更容易。Qt 图形设计器的预览功能可以使开发者能够在开发阶段看到各种样式的图形用户界面，包括客户样式的用户界面。通过 Qt 集成的功能强大的数据库类，Qt 图形设计器还可提供生动的数据库数据浏览和编辑操作。

开发者可以建立同时包含有对话框和主窗口的应用，其中主窗口可以放置菜单，工具栏，旁述帮助等等的子窗口部件。Qt 图形设计器提供了几种表单模板，如果窗体会被多个不同的应用反复使用，那么开发者也可建立自己的表单模板，以确保窗体的一致性。

Qt 图形设计器使用向导来帮助人们更快更方便的建立包含有工具栏、菜单和数据库等方面的应用。程序员可以建立自己的客户窗体，并把它集成到 Qt 图形设计器中。

Qt 图形设计器设计的图形界面以扩展名为“ui”的文件进行保存，这个文件有良好的可读性，这个文件可被 uic（Qt 提供的用户接口编译工具）编译成为 C++的头文件和源文件。Qmake 工具在它为工程生成的 Makefile 文件中自动的包含了 uic 生成头文件和源文件的规则。

另一种可选的做法是，在应用程序运行期间载入 ui 文件，然后把它转变为具备原先全部功能的表单。这样开发者就可以在程序运行期间动态修改应用的界面，而不需重新编译应用，另一方面，也使得应用的文件尺寸减小了。

2.4.3 建立对话框

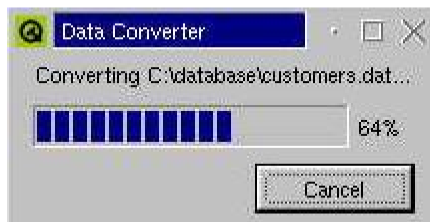
Qt 为许多通用的任务提供了现成的包含了实用的静态函数的对话框类，下边是一些 Qt 的标准的对话框的截屏图。

QmessageBox类是一个用于向用户提供信息或是给用户进行一些简单选择（例如“yes”或“no”）的对话框类。



图十五 一个 QMessageBox 对话框

progressDialog对话框包含了一个进度栏和一个“Cancel”按钮



图十六 一个 QprogressDialog 对话框

Qwizard类提供了一个向导对话框的框架



图十七 一个向导类

Qt提供的对话框还包括QColorDialog，QFileDialog，QFontDialog 和 QprintDialog。这些类通常适用于桌面应用，一般不会在Qt/Embedded中编译使用它们。

2.5 外形与感觉

Qt桌面应用随着执行环境（例如*Windows XP, Mac OS X, Linux*）的不同而具有不同的样式，或者叫做外形与感觉。Qt/Embedded 的应用可以使用不同操作环境提供的样式，也可以以静态或插件的方式使用客户样式。开发者可以设计自己窗体的样式和窗口装饰。

2.5.1 窗体样式

一个样式是一个实现Qt窗体外形与视觉效果的类Qstyle的子类。Qt/Embedded 程

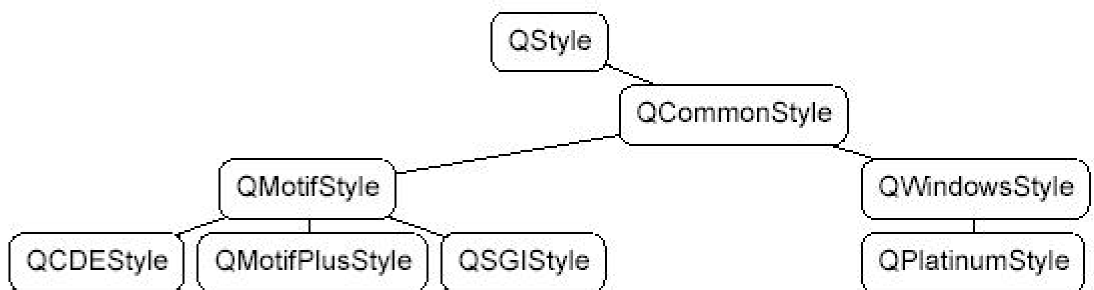
序员可以自由的使用和修改目前存在的样式，也可以使用Qt样式设计引擎实现自己的样式。目前Qt/Embedded提供的样式类型有EmbeddedareWindows, Motif, MotifPlus,CDE, Platinum 和 SGI。样式可以被动态的设置到一个应用中，甚至可以设置到特定的窗体上。



图十八 以不同样式建立的下拉框

通过给一组相关的应用编写一个客户样式，可以让这些应用的外观具有与众不同的感觉。建立客户样式可以通过子类化QStyle，QcommonStyle或者QcommonStyle的派生类来实现。通过重新实现现有样式基类的一两个虚函数可以很容易对现有样式做一些小的修改。

一个样式可以编译成为一个插件，在 Qt 图形设计器中，把样式做成插件的话，开发者就可以以设备的客户样式来预览设计出来的表单。样式插件使得开发者不需重新编译就可改变设备的外观。



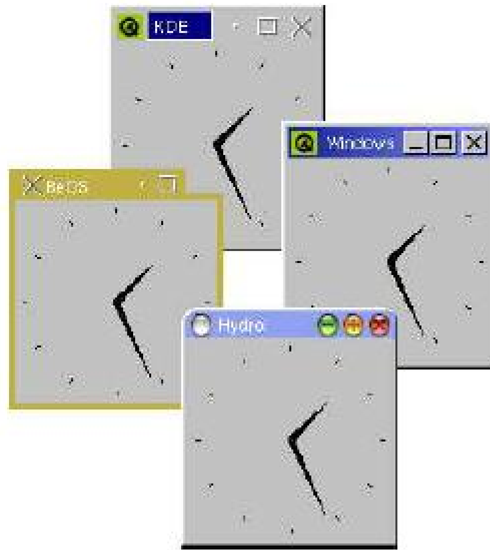
图十九 Qstyle 的继承图

Qt发布的窗体都是样式已知的，当它们的样式改变后，窗体会自动重绘。客户窗

体和对话框大多数都是Qt原先发布的窗体和布局的重新整合。它们也是自动的获得原先的样式。在少数场合，当有必要写一个外观上很随意的窗体时，开发者可以使用QStyle去写一些用户接口元素，而不是直接的绘制固有的矩形。

2.5.2 窗口装饰

顶级窗口的装饰是由一个标题栏和一个框架组成的。Qt/Embedded 包括了这些窗口样式：BeOS, Hydro, KDE 和 Windows。



图二十 不同的平台上的窗口风格

如果需要的话，我们可以配置不同的窗口使用不同的窗口装饰。通过子类化QWSDecoration,我们可以建立客户的装饰样式，并且把它们以插件的形式进行发布。为了在窗口管理器之外进行更多的控制行为，开发者需要子类化QWSManager。

2.6 国际化

Qt/Embedded 完全支持 Unicode，一个国际标准的字符集。开发者在他们的应用中可以自由的混合使用被 Unicode 字符集支持的语言，例如阿拉伯文，英文，中文，希伯来文，日文和俄文等。为了有助于公司将产品推向国际市场，Qt 还提供了将应用翻译成支持多种语言环境的工具。

2.6.1 Unicode

Qt 使用 **QString** 存储 Unicode 编码的字符串，Qstring 取代了粗糙的 const char *；它提供了用于处理 **Qstring** 和 const char *之间相互转换的构造函数和操作符。因为 Qt 使用了隐式共享(写时复制)技术来减少内存的使用,所以直接复制 Qstring 的值是不会产生问题的。为有效率的存储 ASII 码字符串，Qt 还提供了 QCString 类。

Qt为所有要显示在屏幕上的文本，包括最简单的文字标签到最复杂的宽文本编辑器，提供了一个强大的Unicode文本呈现引擎。这个引擎支持一些先进的特征，例如特殊的间隔线、双向写和区别标记。它几乎支持世界上所有的书写系统，包括阿拉伯文，中文，古斯拉夫文，英文，希腊文，希伯来文，日文，韩文，拉丁和越南文。体现这个引擎的最优化性能的常用的例子就是：在带有加速功能的文字的下方显示一条下划线（例如File）。

QtextCodec 的子类用于处理不同编码类型的字符集之间的转换。Qt 3.0 支持 37 种不同的编码方式，包括中文的 Big5 和 GBK,日文的 EUC-JP、JIS 和 Shift-JIS，俄罗斯的 KOI8-R 和 ISO 8859 系列。它们可以以库的一部分或者插件的形式编译，或者使用“feature”机制去除这些编译。

2.6.2 应用的翻译

Qt 提供了相应的工具和函数用于帮助开发者以他们的本地语言推出应用。

要使一个字符串可以被翻译，您需要把这个字符串作为一个参数放到 tr（）函数中调用。例如：

```
saveButton->setText( tr("Save") );
```

Qt 尝试寻找字符串"Save"的译文，如果找到的话，就会把它的译文显示出来，如果找不到，就用原来的字符串"Save"进行显示。例如把英文作为源语言，就需要把这个源语言翻译成中文，即中文为翻译的目标语言，以此类推。这样当调用 tr（）函数后，函数的参数的原先的缺省编码就会转变为 Unicode 编码。Tr()函数的通常用法为：

```
Context::tr("source text", "comment")
```

上面的 “Context” 是指一个 QObject 对象的子类的名称。如果在一个包含了 tr（）成员函数的类的上下文环境中使用 tr（）函数时，“Context”通常可以省略掉。例如：

```
Context：：func1（）
{
    setText( tr("Save") );
}
```

“source text”是指要翻译的文本的内容

“comment”是一个可选项，它用于给手工翻译者提供一些额外的信息。

说了半天，还有一个重要的内容，就是 tr（）函数如何寻找到译文。要把一个源字符串翻译为和它对应的译文（目标语言的字符串）时，需要让 Qt 知道这些译文放在哪里。Qt 规定了译文存储在 QTranslator 对象中，这个对象是从一个内存映射文件（扩展名为 pm）中读取译文。每个 pm 文件包含了某种语言的译文信息。所以开发者需要建立一个 pm 文件来存储应用中需要翻译的字符串的译文。

Qt 提供了 3 种工具帮助人们建立译文存储文件(.pm 文件),这 3 个工具是 lupdate,*Qt Linguist* 和 lrelease.

1）lupdate 自动地从源代码文件（.cpp）和界面接口文件（.ui）中获取所有需要翻译的对象，即上述的（“Context”）；同时还获取要翻译的所有的字符串（源语言的字符串），即上述的“source text”;“comment”选项如果被使用的话，也会被纳入收集的范围。当这些信息收集完毕后，lupdate 最后会生成一个.ts 文件（翻译源文件），这个文件是直接可阅读的。

2)开发者使用 *Qt Linguist 工具提供的良好的人机界面打开*一个.ts 文件(翻译源文件),然后开发者根据每一个 source text 填写上相对应的译文，这样一个.ts 文件（翻译源文件）就包含了完整的“Context”，“source text”和译文信息。

3）最后通过运行 lrelease 去把一个.ts 文件（翻译源文件）压缩为一个.pm 文件（译文存储文件），生成的.pm 文件可用在嵌入式设备上。

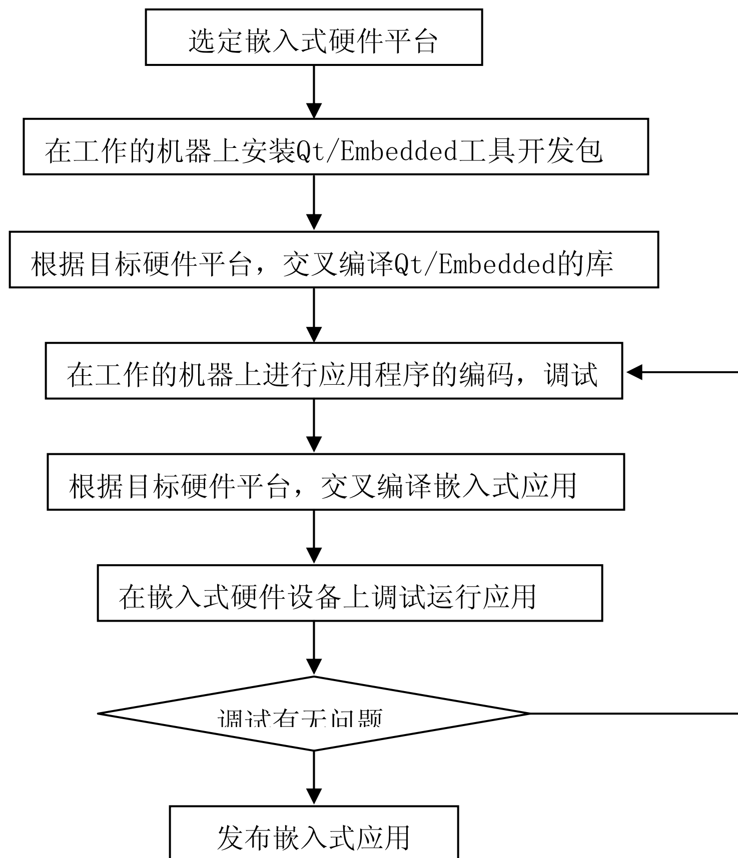
在一个应用的生存期间里，上述的步骤有可能根据需要会被反复执行。多次运行 lupdate 是很安全的，您可以重复使用已经存在的译文（.pm）文件，或者当您不想使用某些翻译文件时你可以标记这些翻译源文件为旧文件，而不需要删除它们。

实 战 篇

前面已经详细介绍了 Qt 嵌入式工具开发包的安装和使用方法，但是这些介绍对于要真正进行一次商业的嵌入式应用开发来说并不足够。嵌入式应用的开发

工作基本上是在工作站或是 PC 机上完成的，在工作的机器上调试运行嵌入式应用,并将输出结果显示在一个仿真小型设备显示终端的模拟器上。在开发的后期，要根据我们选择的嵌入式硬件平台，将嵌入式应用编译链接成适合在这个硬件平台上运行的二进制目标代码，另外由于应用使用到了 Qt/Embedded 的库，所以还要将 Qt/Embedded 库的源代码编译链接成为适合在这个硬件平台上使用的二进制目标代码库。当一个 Qt/Embedded 应用被部署到小型设备上，并可靠的运行，这样一个开发过程才宣告结束。

使用 Qt/Embedded 开发一个嵌入式应用的过程大体可用下面的流程图表示：



图一 Qt/Embedded 应用开发的一般流程

在以下的篇幅，将按照上述的流程完整的介绍使用 Qt/Embedded 开发一个嵌入式应用例子的过程。

1. 嵌入式硬件开发平台的选择

嵌入式系统的核心部件是各种类型的嵌入式处理器，目前据不完全统计，全世界嵌入式处理器的品种总量已经超过 1000 多种，流行体系结构有 30 几个系列，如何在种类纷繁的嵌入式处理器中选择适合应用需求的处理器呢？在应用的需求分析过程中，有几

项因素决定了应该选择什么样的嵌入式处理器：①嵌入式处理器能否在技术上实现应用②嵌入式处理器的成本是否符合应用要求。对于嵌入式处理器能否在技术上实现应用需要考虑到应用在运行时的特点，比如应用对实时性的要求，对计算量和计算速度的要求，对外围接口电路的要求，对图形用户接口的要求等。当选定了一个嵌入式处理器之后，还要考虑选用什么样的操作系统，在选择操作系统时也要注意嵌入式处理器能否被所选的操作系统支持，有没有合适的编译器能够生成支持这种操作系统和嵌入式处理器的二进制目标代码。

近年来，随着 EDI 的推广和 VLSI 设计的普及化，及半导体工艺的迅速发展，在一个硅片上实现一个更为复杂的系统的时代已来临，这就是 System On Chip(SOC)。各种通用处理器内核将作为 SOC 设计公司的标准库，和许多其它嵌入式系统外设一样，成为 VLSI 设计中一种标准的器件，用标准的 VHDL 等语言描述，存储在器件库中。用户只需定义出其整个应用系统，仿真通过后就可以将设计图交给半导体工厂制作样品。这样除个别无法集成的器件以外，整个嵌入式系统大部分均可集成到一块或几块芯片中去，应用系统电路板将变得很简洁，对于减小体积和功耗、提高可靠性非常有利。

SOC 可以分为通用和专用两类。通用系列包括 Infineon 的 TriCore，Motorola 的 M-Core，某些 ARM 系列器件，Echelon 和 Motorola 联合研制的 Neuron 芯片等。专用 SOC 一般专用于某个或某类系统中，不为一般用户所知。一个有代表性的产品是 Philips 的 Smart XA，它将 XA 单片机内核和支持超过 2048 位复杂 RSA 算法的 CCU 单元制作在一块硅片上，形成一个可加载 JAVA 或 C 语言的专用的 SOC，可用于公众互联网如 Internet 安全方面。

由于 SOC 芯片集成了处理器和许多常用的外围接口芯片，使得它的功能变得很强大，能够应用的领域和场合变得很广，所以嵌入式开发板的厂商选择 SOC 芯片作为开发板的

核心芯片。三星的 S3C2440 就是一款带有 ARM920T 处理器内核的 SOC 芯片，由于它主频高，内置 LCD 和触摸屏控制器，以及声音控制器等外围电路，因而用在对图形用户接口有较高要求的场合是非常合适的。因为支持 ARM9 处理器的 linux 编译器早已发布，所以 S3C2440 可以很好的支持 linux 和 Qt/Embedded 的运行。

深圳优龙科技有限公司设计开发的 YL2440 嵌入式开发板是一个成熟稳定的嵌入式开发硬件平台，它使用了三星的 S3C2440 作为核心芯片。以下是 YL2440 的一些资料（更详细的资料可参考优龙公司的《YL2440 使用手册》）。在本文提及的 Qt/Embedded 应用例子都是居于 YL2440/YL2410 嵌入式硬件开发平台的。

2. 安装 Qt/Embedded 工具开发包

要进行 Qt/Embedded 开发，就需要在工作的机器上安装 Qt/Embedded 工具开发包，这一步已在《Qt 嵌入式开发（入门篇)》的第一节中谈及，在此不在敖述。

3. 交叉编译 Qt/Embedded 的库

开发居于 Qt/Embedded 的应用程序要使用到 Qt/Embedded 的库，编写的 Qt 嵌入式应用程序最终是在 YL2440
开发板上运行的，因此在把 Qt 嵌入式应用程序编译成支持 YL2440 的目标代码之前，需要两样东西，一个是 arm9 的 linux 编译器，另一个是经 arm9 的 linux 编译器编译过的 Qt/Embedded 的库。

①安装交叉编译工具

需要arm9的linux编译器去编译工程并产生arm9处理器的目标代码，却是在一台PC机或者工作站上（这个机器有可能是x86的处理器）使用这个编译器，这个过程被称为交叉编译。交叉编译的实际定义是在一个处理器平台上编译产生一个工程代码的另

一个处理器的目标代码。关于如何安装一个arm9的linux编译器请参考第一章节,但是需要特别提醒读者注意，使用toolchain做为交叉编译工具时，最好使用cross-3.3.2及其以后的版本，这样才能对qt/embeded有良好支持。

②交叉编译Qt/Embedded库

当有了arm9的linux编译器之后，就可以使用这个编译器交叉编译Qt/Embedded库的源代码，从而产生一

个以ARM9为目标代码的Qt/Embedded库。具体过程如下

1、 解包 Qt/Embedded（以 Qt/Embedded2.3.7 为例）

解包这个 Qt/Embedded2.3.7 压缩包时，应把它解压缩到不同于您的机器的处理器使用的 Qt/Embedded2.3.7 的安

装路径。

在 Linux 命令模式下运行以下命令：

```
tar xfz qt-embedded-2.3.7.tar.gz
```

2、配置 Qt/Embedded2.3.7 的安装

Qt/Embedded 的安装选项有很多个，您可以在命令行下直接输入“./configure”来运行配置，这时安装程序会一步一步提示你输入安装选项。您也可以在“./configure”后输入多个安装选项直接完成安装的配置。在这些选项中有一个选项决定了编译 Qt/Embedded 库的范围，即可以指定以最小，小，中，大，完全 5 种方式编译 Qt/Embedded 库。另外 Qt/Embedded 的安装选项还允许我们自己定制一个配置文件，来有选择的编译 Qt/Embedded 库，这个安装选项是“-qconfig local”；当我们指定这个选项时，Qt/Embedded 库在安装过程中会寻找 qt-2.3.7/src/tools/qconfig-local.h 这个文件，

如找到这个文件，就会以该文件里面定义的宏，来编译链接 Qt/Embedded 库。

由于使用优龙公司的 YL2440 嵌入式开发板，所以为了支持触摸屏的显示，需要定义一些宏，为了避免初学者在一开始就直接接触到 Qt/Embedde 的复杂的宏编译选项，把这些宏定义到一个名为 qconfig-local.h 的安装配置文件中，当使用者在安装 Qt/Embedded 的时候，需要把这个文件复制到 Qt/Embedded 的安装路径的/src/tools 子路径下。例如您安装的是 Qt/Embedded2.3.7，那么您应该把提供给您的 qconfig-local.h 安装配置文件复制到 qt-2.3.7/src/tools 路径下。

具体过程如下：

```
cd qt-2.3.7
export QTDIR=$PWD
export QTEDIR=$QTDIR
cp /配置文件所在路径/qconfig-local.h   ./src/tools
make clean
./configure  - xplatform linux-arm-g++ -shared  - debug  （接下行）
-qconfig local -qvfb -depths 4,8,16,32
make
cd ..
```

4. 一个 Hello,World 的例子

下面将通过编写一个跳动的“Hello,World”字符串，来讲解 Qt 嵌入式应用的开发过程。

① 生成一个工程文件（.pro 文件）

一个应用通常对应一个工程文件，生成一个工程文件，并对它做一些简单的编辑，然后

使用一个专门的工具（例如 tmake）处理这个工程文件，就可以生成一个 Makefile 文件。

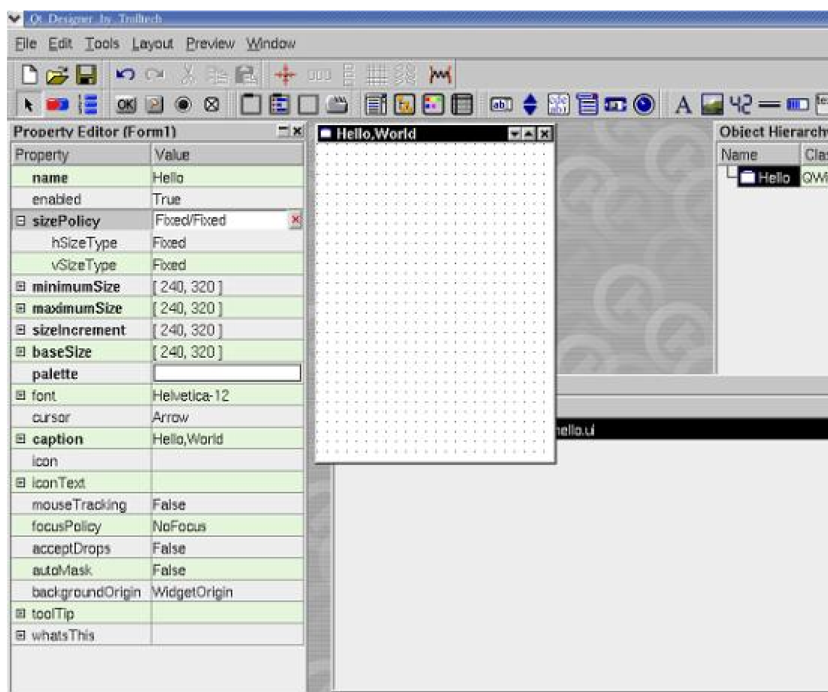
产生一个工程文件的其中一个方法是使用 progen 命令(progen 程序可在 tmake 的安装路径下找到)。下面使用 progen 产生一个名为 hello 的工程文件

```
progen  -t app.t  -o hello.pro
```

产生的 hello.pro 工程文件并不完整，开发者还需手动添加工程所包含的头文件，源文件等信息。

② 新建一个窗体

在 qt-2.3.x for x11 的安装路径的 bin 目录下运行“./designer”命令，就启动了一个 Qt 图形编辑器。点击编辑器的“new”菜单，弹出了一个“new Form”对话框，在这个对话框里选择“Widget”，然后点击“OK”按钮，这样就新建了一个窗体。接着对这个窗体的属性进行设置，注意把窗体的“name”属性设为“Hello”；窗体的各种尺寸设为宽“240”，高“320”，目的是使窗体大小和 YL2440 带的显示屏大小一致；窗体背景颜色设置为白色。具体设置可参见下图：



图二　Hello 窗体的属性设置

设置完成后，将其保存为 hello.ui 文件,这个文件就是 Hello 窗体的界面存储文件。

③ 生成 Hello 窗体类的头文件和实现文件

下面根据上述的界面文件 hello.ui 使用 uic 工具产生出 Hello 窗体类的头文件和实现文件，具体方法是：

```
cd qt-2.3.7/bin
uic  -o hello.h  hello.ui
uic  -o hello.cpp  -impl hello.h  hello.ui
```

这样就得到了 Hello 窗体类的头文件 hello.h 和实现文件 hello.cpp。下面就可以根据我们要实现的具体功能，在 hello.cpp 文件里添加相应的代码。

比如要在 Hello 的窗体上显示一个动态的字符串“Hello，World”，那么需要重新实现 paintEvent(QPaintEvent *)方法，同时还需要添加一个定时器 Qtimer 实例，以周期性刷新屏幕，从而得到动画得效果。下面是修改后的 hello.h 和 hello.cpp 文件。

```
/****************************************************************************
*
**  以下是 hello.h 的代码
****************************************************************************
/
#ifndef HELLO_H
#define HELLO_H

#include <qvariant.h>
```

```
#include <qwidget.h>
class QVBoxLayout;
class QHBoxLayout;
class QGridLayout;

class Hello : public QWidget
{
    Q_OBJECT

public:
    Hello( QWidget* parent = 0, const char* name = 0, WFlags fl = 0 );
    ~Hello();

//以下是手动添加的代码
signals:
    void clicked();
protected:
    void mouseReleaseEvent( QMouseEvent * );
    void paintEvent( QPaintEvent * );
private slots:
    void animate();
private:
    QString t;
    int     b;
};
#endif // HELLO_H
/****************************************************************************
*
**  以下是 hello.cpp 源代码
****************************************************************************
/

#include "hello.h"
```

```
#include <qlayout.h>
#include <qvariant.h>
#include <qtooltip.h>
#include <qwhatsthis.h>
#include <qpushbutton.h>
#include <qtimer.h>
#include <qpainter.h>
#include <qpixmap.h>

/*
 *  Constructs a Hello which is a child of 'parent', with the
 *  name 'name' and widget flags set to 'f'
 */
Hello::Hello( QWidget* parent,  const char* name, WFlags fl )
    : QWidget( parent, name, fl )
{
    if ( !name )
    setName( "Hello" );
    resize( 240, 320 );
    setMinimumSize( QSize( 240, 320 ) );
    setMaximumSize( QSize( 240, 320 ) );
    setSizeIncrement( QSize( 240, 320 ) );
    setBaseSize( QSize( 240, 320 ) );
    QPalette pal;
    QColorGroup cg;
    cg.setColor( QColorGroup::Foreground, black );
    cg.setColor( QColorGroup::Button, QColor( 192, 192, 192) );
    cg.setColor( QColorGroup::Light, white );
    cg.setColor( QColorGroup::Midlight, QColor( 223, 223, 223) );
    cg.setColor( QColorGroup::Dark, QColor( 96, 96, 96) );
    cg.setColor( QColorGroup::Mid, QColor( 128, 128, 128) );
    cg.setColor( QColorGroup::Text, black );
```

```
cg.setColor( QColorGroup::BrightText, white );
cg.setColor( QColorGroup::ButtonText, black );
cg.setColor( QColorGroup::Base, white );
cg.setColor( QColorGroup::Background, white );
cg.setColor( QColorGroup::Shadow, black );
cg.setColor( QColorGroup::Highlight, black );
cg.setColor( QColorGroup::HighlightedText, white );
pal.setActive( cg );
cg.setColor( QColorGroup::Foreground, black );
cg.setColor( QColorGroup::Button, QColor( 192, 192, 192) );
cg.setColor( QColorGroup::Light, white );
cg.setColor( QColorGroup::Midlight, QColor( 220, 220, 220) );
cg.setColor( QColorGroup::Dark, QColor( 96, 96, 96) );
cg.setColor( QColorGroup::Mid, QColor( 128, 128, 128) );
cg.setColor( QColorGroup::Text, black );
cg.setColor( QColorGroup::BrightText, white );
cg.setColor( QColorGroup::ButtonText, black );
cg.setColor( QColorGroup::Base, white );
cg.setColor( QColorGroup::Background, white );
cg.setColor( QColorGroup::Shadow, black );
cg.setColor( QColorGroup::Highlight, black );
cg.setColor( QColorGroup::HighlightedText, white );
pal.setInactive( cg );
cg.setColor( QColorGroup::Foreground, QColor( 128, 128, 128) );
cg.setColor( QColorGroup::Button, QColor( 192, 192, 192) );
cg.setColor( QColorGroup::Light, white );
cg.setColor( QColorGroup::Midlight, QColor( 220, 220, 220) );
cg.setColor( QColorGroup::Dark, QColor( 96, 96, 96) );
cg.setColor( QColorGroup::Mid, QColor( 128, 128, 128) );
cg.setColor( QColorGroup::Text, black );
cg.setColor( QColorGroup::BrightText, white );
cg.setColor( QColorGroup::ButtonText, QColor( 128, 128, 128) );
cg.setColor( QColorGroup::Base, white );
```

```
    cg.setColor( QColorGroup::Background, white );
    cg.setColor( QColorGroup::Shadow, black );
    cg.setColor( QColorGroup::Highlight, black );
    cg.setColor( QColorGroup::HighlightedText, white );
    pal.setDisabled( cg );
    setPalette( pal );
    QFont f( font() );
    f.setFamily( "adobe-helvetica" );
    f.setPointSize( 29 );
    f.setBold( TRUE );
    setFont( f );
    setCaption( tr( "" ) );

//以下是手动添加的代码
    t = "Hello,World";
    b = 0;
    QTimer *timer = new QTimer(this);
    connect( timer, SIGNAL(timeout()), SLOT(animate()) );
    timer->start( 40 );
}
/*
 *  Destroys the object and frees any allocated resources
 */
Hello::~Hello()
{
}
/*
  This private slot is called each time the timer fires.
*/

//以下至结尾是手动添加的代码
void Hello::animate()
{
```

```
    b = (b + 1) & 15;
    repaint( FALSE );
}



/*
  Handles mouse button release events for the Hello widget.

  We emit the clicked() signal when the mouse is released inside
  the widget.
*/

void Hello::mouseReleaseEvent( QMouseEvent *e )
{
    if ( rect().contains( e->pos() ) )
        emit clicked();
}




/*  Handles paint events for the Hello widget.

  Flicker-free update. The text is first drawn in the pixmap and the
  pixmap is then blt'ed to the screen.
*/

void Hello::paintEvent( QPaintEvent * )
{
    static int sin_tbl[16] = {
        0, 38, 71, 92, 100, 92, 71, 38, 0, -38, -71, -92, -100, -92, -71, -38};
    if ( t.isEmpty() )
        return;
    // 1: Compute some sizes, positions etc.
```

```
    QFontMetrics fm = fontMetrics();
    int w = fm.width(t) + 20;
    int h = fm.height() * 2;
    int pmx = width()/2 - w/2;
    int pmy = height()/2 - h/2;

    // 2: Create the pixmap and fill it with the widget's background
    QPixmap pm( w, h );
    pm.fill( this, pmx, pmy );

    // 3: Paint the pixmap. Cool wave effect
    QPainter p;
    int x = 10;
    int y = h/2 + fm.descent();
    int i = 0;
    p.begin( &pm );
    p.setFont( font() );
    while ( !t[i].isNull() ) {
        int i16 = (b+i) & 15;
        p.setPen( QColor((15-i16)*16,255,255,QColor::Hsv) );
        p.drawText( x, y-sin_tbl[i16]*h/800, t.mid(i,1), 1 );
        x += fm.width( t[i] );
        i++;
    }
    p.end();
    // 4: Copy the pixmap to the Hello widget
    bitBlt( this, pmx, pmy, &pm );
}
```

④ 编写主函数 main()

一个 Qt/Embeded 应用程序应该包含一个主函数，主函数所在的文件名是 main.cpp。主函数是应用程序执行的入口点。以下是 Hello,World 例子的主函数文件 main.cpp 的实现代码。

```
/***************************************************************************
*
**  以下是 main.cpp 源代码
***************************************************************************
/
#include "hello.h"
#include <qapplication.h>


/*
  The program starts here. It parses the command line and builds a message
  string to be displayed by the Hello widget.
*/
#define QT_NO_WIZARD

int main( int argc, char **argv )
{
    QApplication a(argc,argv);
    Hello dlg;
    QObject::connect( &dlg, SIGNAL(clicked()), &a, SLOT(quit()) );
    a.setMainWidget( &dlg );
    dlg.show();
    return a.exec();
}
```

⑤ 编辑工程文件 hello.pro 文件

到目前为止，为 Hello,World 例子编写了一个头文件和两个源文件，这 3 个文件应该被包括在工程文件中，因此还需要编辑 hello.pro 文件，加入这 hello.h,hello.cpp,main.cpp 这三个文件名。具体定义如下

```
/*************************************************************************
*
**  以下是 hello.pro 文件的内容
*************************************************************************
/

TEMPLATE    = app
CONFIG      = qt warn_on release
HEADERS     = hello.h
SOURCES     = hello.cpp \
              main.cpp
INTERFACES  =
```

⑥ 生成 Makefile 文件

编译器是根据 Makefile 文件内容来进行编译的，所以需要生成 Makefile 文件。Qt 提供的 tmake 工具可以帮助我们从一个工程文件（.pro 文件）中产生 Makefile 文件。结合当前例子，要从 hello.pro 生成一个 Makefile 文件的做法是：首先查看环境变量 $TMAKEPATH 是否指向 arm 编译器的配置目录，在命令行下输入以下命令

```
echo $TMAKEPATH
```

如果返回的结果的末尾不是 。。。。/qws/linux-arm-g++ 的字符串，那您需要把环境变量 $TMAKEPATH 所指的目录设置为指向 arm 编译器的配置目录，过程如下，

```
export TMAKEPATH = /tmake 安装路径/qws/linux-arm-g++
```

同时，应确保当前的 QTDIR 环境变量指向 Qt/Embedded 的安装路径,如果不是，则需要执行以下过程

```
export QTDIR = ……/qt-2.3.7
```

上述步骤完成后，就可以使用 tmake 生成 Makefile 文件，具体做法是在命令行输入以下命令：

```
tmake －o Makefile hello.pro
```

这样就可以看到当前目录下新生成了一个名为 Makefile 的文件。下一步，需要打开这个文件，做一些小的修改。

（1）将 LINK = arm-linux-gcc 这句话改为

LINK = arm-linux-g++

这样做是因为要是用 arm-linux-g++进行链接

（2）将 LIBS = $(SUBLIBS) -L$(QTDIR)/lib -lm －lqte 这句话改为

LIBS = $(SUBLIBS) -L/usr/local/arm/2.95.3/lib -L$(QTDIR)/lib -lm －lqte

这是因为链接时要用到交叉编译工具 toolchain 的库。

⑦ 编译链接整个工程

最后就可以在命令行下输入 make 命令对整个工程进行编译链接了。

```
make
```

make 生成的二进制文件 hello 就是可以在 YL2440 上运行的可执行文件。下面将介绍如何将这个文件发布到 YL2440 上。

5. 发布一个 Qt/Embeded 应用到 YL2440

在优龙公司的《YL2440 使用手册》的第三章“Linux 的烧写和引导”介绍了如何把 Linux 的引导和内核的映象文件烧写到 YL2440 的 FLASH 上。一个 Qt/Embedded 应用的运行需要有 Linux 操作系统和 Qt/Embedded 库的支持。所以我们除了要烧写 Linux 到 YL2440 的 FLASH 存储空间之外，我们还要烧写 Qt/Embedded 的二进制库到 YL2440 的 FLASH。一般来说，先把 Qt/Embedded 的二进制库复制到某个目录下，然后再把这个目录制成某种类型的根文件系统，最后把这个根文件系统烧写到 YL2440 的 FLASH 上，这个过程可能需要一些制作根文件系统的工具，例如 mkcramfs。在优龙公司的 YL2440 开发套件中附有的一张光盘中，有一个叫 Linux 的文件夹，包含了一个名为 qtopia.cramfs 的根文件系统映象，这个根文件系统包含了 trolltech 公司使用 Qt/Embedded 开发的一个 PDA 应用环境（简称 QPE），当用户把这个根文件系统映象烧写到 YL2440 的 FLASH 存储空间后，在 YL2440 的根文件系统下就包含了一个 Qt/Embedded 2.3.7 版本的二进制库文件。

鉴于目前发行的套件中所安装的 arm linux 均采用只读文件系统作为其根文件系统，因此其目录大多是不可写的。只有/var，/tmp 是 RAM 盘可写，但板子一掉电里面的内容就丢失了，因此只能作临时文件保存，无法永久的保存数据，例如配置文件等。但是/var，/tmp 这些目录可作为调试程序的目录，可以把编译好的 Qt/Embedded 应用程序传送到这些目录，并运行。

下面简单介绍一下如何把一个 Qt/Embedded 应用程序传送到 arm linux 的根文件系统的/tmp 目录。

① 用 YL2440 开发套间中的串口线连接好开发用的机器的串口和 YL2440 开发板的串口。
② 在 WINDOWS 操作系统的程序|附件|通讯中运行一个超级终端，并设定通过串口 x 与 YL2440 进行连接。具体设置过程见下列图示：



图三 输入连接名称

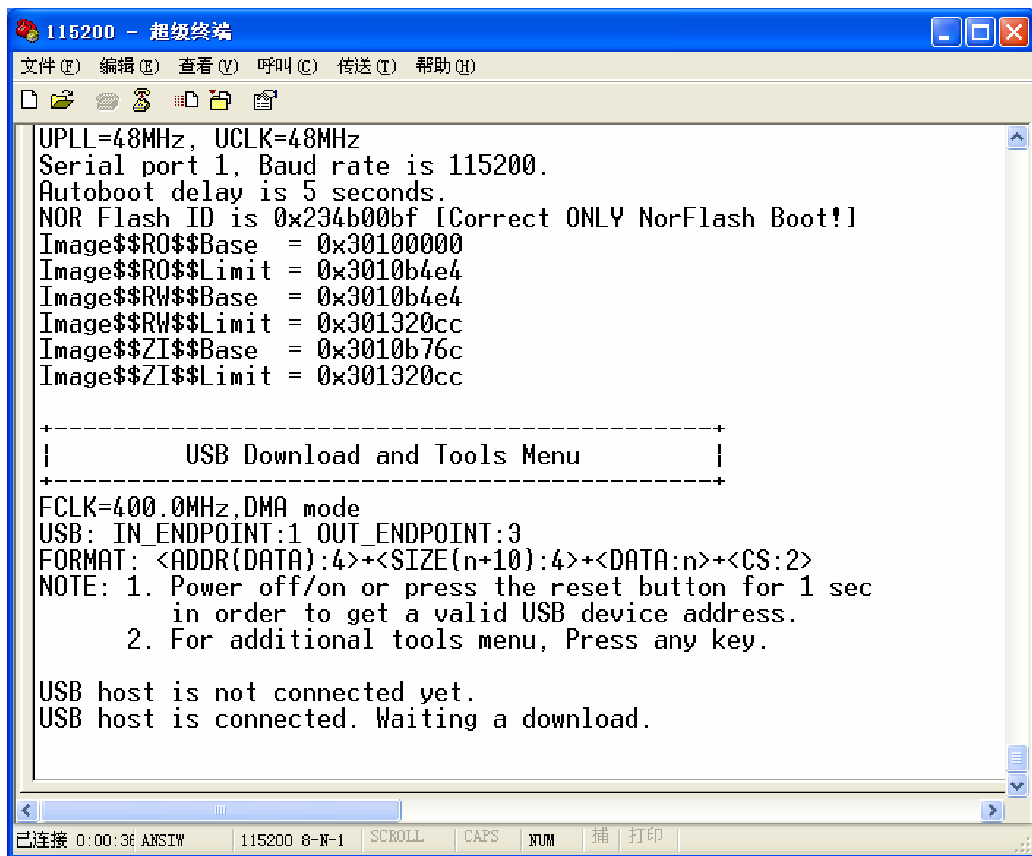


图四 输入连接所使用的串口的端口号



图五 设置串口通讯的的参数

③ 打开 YL2440 的电源，或者复位 YL2440，这时可在超级终端的窗口中看到 BIOS 引导信息，如下图



④ 待 Linux 引导完成后，在当前文件系统的根目录下，运行“cd tmp”命令，进入到 tmp 目录，如下图

1323 - 超级终端

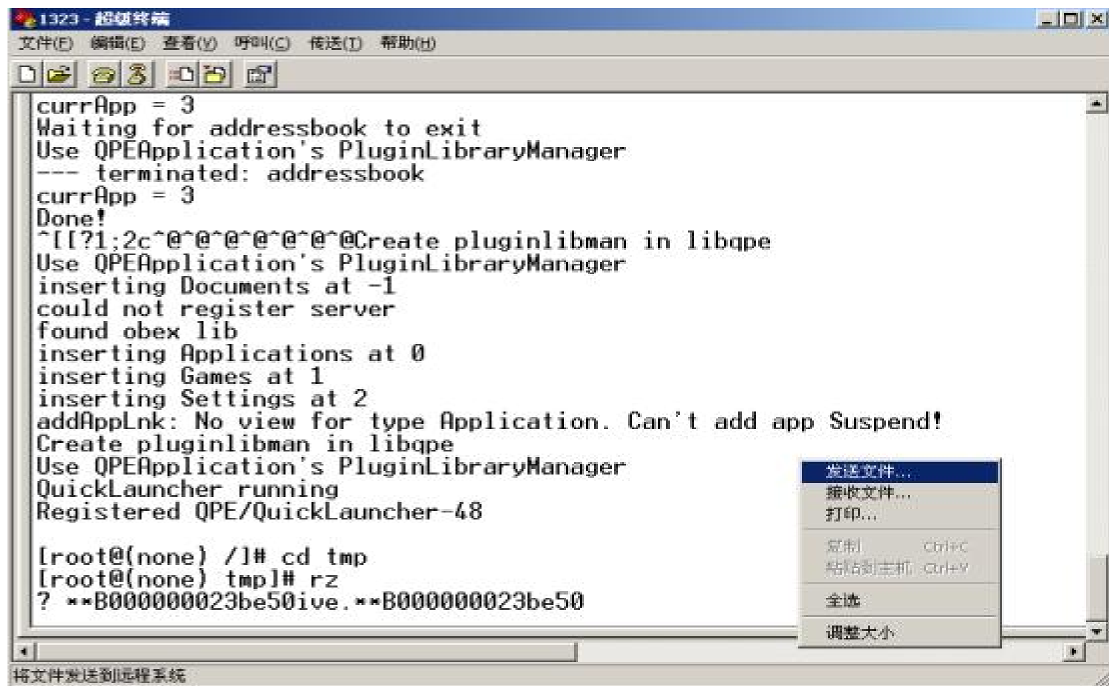
文件(F) 编辑(E) 查看(V) 呼叫(C) 传送(T) 帮助(H)

```
Create pluginlibman in libqpe
currApp = 3
Waiting for addressbook to exit
Use QPEApplication's PluginLibraryManager
--- terminated: addressbook
currApp = 3
Done!
^[[?1;2c^@^@^@^@^@^@^@Create pluginlibman in libqpe
Use QPEApplication's PluginLibraryManager
inserting Documents at -1
could not register server
found obex lib
inserting Applications at 0
inserting Games at 1
inserting Settings at 2
addAppLnk: No view for type Application. Can't add app Suspend!
Create pluginlibman in libqpe
Use QPEApplication's PluginLibraryManager
QuickLauncher running
Registered QPE/QuickLauncher-48

[root@(none) /]# cd tmp
[root@(none) tmp]#
```

连接的 0:01:40 VT100J 115200 8-N-1 SCROLL CAPS NUM 捕 打印

⑤ 在当前命令行下输入“rz”命令，开始一个文件传输过程；接着点击鼠标右键，在弹出的菜单中，点击“发送文件”子菜单，然后选择你要传送到 tmp 目录的 Qt/Embedded 应用程序。



例如，下图是正在传送 Hello,World 例子的可执行文件到 tmp 目录



⑥ 最后就可以在 tmp 目录下运行传送过来的一个 Qt/Embedded 应用程序。

6. 添加一个 Qt/Embedded 应用到 QPE

前面提到了优龙光盘里面有一个名为 qtopia.cramfs 的根文件系统映象，烧写这个根文件系统到 YL2440 的 FLASH，就会安装有 QT PDA 应用环境（简称 QPE）。可以在 QPE 里添加我们自己编写的应用，不过添加的过程需要重新交叉编译 QPE 的源文件。下面我们介绍如何在 QPE 里添加我们编写的 Hello，World 的例子。

① 在工作的机器上解包 qtopia

```
tar zxvf qtopia-free-1.7.x.tar.gz
cd qtopia-free-1.7.x
export QTDIR=$QTEDIR
export QPEDIR=$PWD
export PATH=$QPEDIR/bin:$PATH
```

注意在上面已经设定环境变量 QPEDIR 为 QPE 的安装（解包）路径。

② 建立 Hello，World 的例子程序的图标文件

方法是：制作一个 32 X 32 大小的 PNG 格式的图标文件，将该文件存放在$QPEDIR/pics/inline 目录下，然后使用以下命令将$QPEDIR/pics/inline 目录下的所有图形文件转换成为一个 c 语言的头文件，这个头文件包含了该目录下的图形文件的 rgb 信息。

```
qembed --images $QPEDIR/pics/inline/*.*
        > $QPEDIR/src/libraries/qtopia/inlinepics_p.h
```

注意上述的 qembed 是在 qt3.x.x for x11 上才发布有的工具

③ 交叉编译 qtopia

在$QPEDIR 路径下，运行以下命令

```
cd src
./configure -platform linux-arm-g++
make
cd ..
```

④ 建立应用启动器（.desktop）文件

方法是：建立一个文本文件，在文件中添加以下的内容，这些内容指明了应用的名称，图标名等信息，然后将文件更名为 xxxx.desktop,保存在$QPEDIR/apps/applications 目录下。

以下是例子程序的启动器文件（hello.desktop）：

```
[Desktop Entry]
Comment=A Hello Program
Exec=hello
Icon=Hello
Type=Application
Name=hello
```

① 建立根文件系统

在这里利用原有的 qtopia.cramfs 的根文件系统映象，把新建的应用的相关文件添加到这个根文件系统中。首先要把 qtopia.cramfs 的根文件系统 mount 到工作机器上来，然后复制这个文件系统的内容到一个临时的 temp 目录，这时可以在 temp\Qtopia 目录下看到一个 qtopia-free-1.7.0 目录，这就是 qtopia.cramfs 的根文件系统里的 qpe 安装目录，接着把新建的应用的相关文件（包括启动器文件，包含了图标的库文件

libqte.so.*，和应用程序的可执行文件）复制到 temp/Qtopia/ qtopia-free-1.7.0 的对应的目录。具体过程如下

```
mkdir /mnt/cram
mount －t cramfs  qtopia.cramfs  /mnt/cram －o loop
cp －ra  /mnt/cram  temp/
cp $QPEDIR/apps/Applications/hello.desktop  （接下行）
           temp/Qtopia/qtopia-free-1.7.0/apps/Applications/
cp $QPEDIR/lib/libqpe.so.*  （接下行）
           temp/Qtopia/qtopia-free-1.7.0/lib/
cp hello  temp/Qtopia/qtopia-free-1.7.0/bin （拷贝可执行文件）
mkcramfs temp xxxxxx.cramfs         (生成新的根文件系统)
```

将生成的新的根文件系统烧写到 YL2440 的 FLASH 根文件系统区，复位，OK,就可以看到 QPE 里有我们编写的应用的图标了，点击这个图标，程序就成功运行了。当然为了使我们的应用和原先的 QPE 的应用具备统一的界面风格，在编写自己的应用的主函数文件（main.cpp）时，不妨使用 QPE 提供的宏，具体可参考 QPE 应用程序的源文件。下面是我们编写的 Hello,world 程序在 qvfb 的截屏图。



小记

开发 Qt/Embedded 应用对初学者可能是一项艰苦的工作过程，因为需要安装和设置很多的内容，有时候某一过程没有进行可能会导致一些莫名奇妙的出错提示。尽管我们的开发文档详细的介绍了嵌入式 Qt 的开发过程，然而还是不能保证初学者一步一步按照我们所描述的去做便可以在编译应用时万无一失，因为 linux 开发包之间有一定的依赖性，这些开发包又从属不同的开发商或组织。我们的建议是您在您的机器上安装 linux 并准备进行开发时，至少要安装一个工作站版的 linux。您需要注意您的机器上安装有 libjpeg，e2fsprogs，

Freetypes 等开发包有否安装，因为上述谈到的 Qt 嵌入式的软件都用到了这些开发包。

您不必为这些繁琐的工作感到沮丧，因为这些工作会让你有机会认识嵌入式软件工作的底层细节。您会把嵌入式软件的开发过程当作是在玩一个锻炼智力的玩具吗？您很快会有喜欢上这种玩具的感觉的！

附录二 超级终端的设置

超级终端的设置请看光盘的“多媒体演示”目录下的“超级终端的配置.exe”文件。

附录三 DNW 的设置

DNW 的使用请查看光盘的“多媒体演示”目录下的“DNW 串口工具配置.exe”文件。

附录四 常见问答

1. LINUX 部份

1.1 LINUX 下如何加载 U 盘

1) LINUX 操作系统运行后，将 U 盘插入 USB HOST0 插口，超级终端会显示下图所示信息

```
          RX bytes:2350 (2.2 kiB)  TX bytes:0 (0.0 iB)
          Interrupt:37 Base address:0x300

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 iB)  TX bytes:0 (0.0 iB)

[root@(none) /]# lsmod
Module                  Size  Used by    Not tainted
s3c2410-uda1341         9944   0
[root@(none) /]# hub.c: USB new device connect on bus1/1, assigned device number
 2
usb.c: USB device not accepting new address=2 (error=-110)
hub.c: USB new device connect on bus1/1, assigned device number 3
usb.c: USB device not accepting new address=3 (error=-110)
hub.c: USB new device connect on bus1/1, assigned device number 4
usb.c: USB device 4 (vend/prod 0xea0/0x6803) is not claimed by any active driver
.

[root@(none) /]# _
```

若显示出“(vend/prod xxxx/xxxx)”说明此设备已与 USB HOST 驱动建立通讯，如显示“is not claimed by any active driver”说明尚未找到此类设备对应的驱动程序。

2) 以模块方式添加 U 盘驱动程序，u 盘的驱动需要两个模块的支持，分别是 usb-storage.o 和 sd_mod.o，它们都是在编译内核源代码包后生成的。这两个模块的安装方法是：在命令行下执行 insmod usb-storage.o 和 insmod sd_mod.o 命令，

```
 2
usb.c: USB device not accepting new address=2 (error=-110)
hub.c: USB new device connect on bus1/1, assigned device number 3
usb.c: USB device not accepting new address=3 (error=-110)
hub.c: USB new device connect on bus1/1, assigned device number 4
usb.c: USB device 4 (vend/prod 0xea0/0x6803) is not claimed by any active driver
.

[root@(none) /]# insmod usb-storage
Initializing USB Mass Storage driver...
usb.c: registered new driver usb-storage
scsi0 : SCSI emulation for USB Mass Storage devices
  Vendor: SAMSUNG   Model: SAMSUNGDISK        Rev: 1.11
  Type:   Direct-Access                       ANSI SCSI revision: 02
USB Mass Storage support registered.
[root@(none) /]# insmod sd_mod
Attached scsi removable disk sda at scsi0, channel 0, id 0, lun 0
SCSI device sda: 63232 512-byte hdwr sectors (32 MB)
sda: Write Protect is off
Partition check:
 /dev/scsi/host0/bus0/target0/lun0: p1
[root@(none) /]# ls /dev/scsi/host0/bus0/target0/lun0/
disc  part1
[root@(none) /]#
```

加载模块后会出现一些提示信息，有“sda: Write Protect……”，“Partiyion check……”等信息后表示此驱动已识别并可读写 U 盘了，此后可查看/dev/scsi/目录下会自动建立相关目录和设备文件。

3) 挂载 U 盘上的文件系统，假设我们要将 U 盘挂载到/tmp/udisk 目录，那么我们先确认 tmp 下已经存在 udisk 这个目录，然后再执行 mount 操作，

```
[root@(none) mnt]# insmod sd_mod
Attached scsi removable disk sda at scsi0, channel 0, id 0, lun 0
SCSI device sda: 63232 512-byte hdwr sectors (32 MB)
sda: Write Protect is off
 /dev/scsi/host0/bus0/target0/lun0: p1
[root@(none) mnt]# mount
/dev/mtdblock/2 on / type cramfs (rw)
none on /dev type devfs (rw)
ramfs on /var type ramfs (rw)
none on /proc type proc (rw)
none on /dev/pts type devpts (rw)
tmpfs on /dev/shm type tmpfs (rw)
[root@(none) mnt]# mount /dev/scsi/host0/bus0/target0/lun0/part1 udisk/
VFS: Can't find ext2 filesystem on dev sd(8,1).
cramfs: wrong magic
[root@(none) mnt]# mount
/dev/mtdblock/2 on / type cramfs (rw)
none on /dev type devfs (rw)
ramfs on /var type ramfs (rw)
none on /proc type proc (rw)
none on /dev/pts type devpts (rw)
tmpfs on /dev/shm type tmpfs (rw)
/dev/scsi/host0/bus0/target0/lun0/part1 on /mnt/udisk type msdos (rw)
[root@(none) mnt]# _
```

方法是在命令行下敲以下命令：

mount －t vfat /dev/scsi/host0/bus0/target0/lun0/part1 /tmp/udisk

将 U 盘挂载到 udisk 目录后，再用 mount 命令查看系统已挂载的文件系统，其中“/dev/scsi/xxxx on /tmp/udisk type msdos ”表明在 udisk 目录下挂载 msdos 文件系统成功。另外，由于 usb 设备的每个根端口有 4 个 HUB 子端口，对应这四个端口有 4 个驱动设备文件，它们位于/dev/scsi/host0/bus0/target0/lun0/目录下，名称分别为 part1, part2, part3, part4.如果挂载 U 盘命令执行不成功，有可能是选择的驱动文件不对，这时可尝试使用 part2,part3,part4 驱动文件挂载 U 盘，例如

```
mount -t vfat /dev/scsi/host0/bus0/target0/lun0/part2 /tmp/udisk
mount -t vfat /dev/scsi/host0/bus0/target0/lun0/part3 /tmp/udisk
mount -t vfat /dev/scsi/host0/bus0/target0/lun0/part4 /tmp/udisk
```

4) 查看 U 盘内容，进入 udisk 目录，执行 ls 命令

```
[root@(none) mnt]# mount
/dev/mtdblock/2 on / type cramfs (rw)
none on /dev type devfs (rw)
ramfs on /var type ramfs (rw)
none on /proc type proc (rw)
none on /dev/pts type devpts (rw)
tmpfs on /dev/shm type tmpfs (rw)
[root@(none) mnt]# mount /dev/scsi/host0/bus0/target0/lun0/part1 udisk/
VFS: Can't find ext2 filesystem on dev sd(8,1).
cramfs: wrong magic
[root@(none) mnt]# mount
/dev/mtdblock/2 on / type cramfs (rw)
none on /dev type devfs (rw)
ramfs on /var type ramfs (rw)
none on /proc type proc (rw)
none on /dev/pts type devpts (rw)
tmpfs on /dev/shm type tmpfs (rw)
/dev/scsi/host0/bus0/target0/lun0/part1 on /mnt/udisk type msdos (rw)
[root@(none) mnt]#
[root@(none) mnt]# cd udisk/
[root@(none) udisk]# ls
demo????  instal~1.exe  lcd_lt~1.h  readme.txt  zimage        ??????~1.doc
i2stest   lcd_lt~1.c    matrix.mpg  win2k_xp    ??????~1.mp3  ????
[root@(none) udisk]#
```

以后可以象 PC 机访问硬盘一样在 udisk 目录下对 U 盘进行读写操作。

5) 2440 卸载 U 盘的方法

卸载 U 盘前要确保你的 linux 终端当前并不在 u 盘的 mount 的目标文件夹内，否则会提示

“设备忙”这样的字眼。卸载 U 盘的命令行操作如下

```
umount /dev/scsi/host0/bus0/target0/lun0/part1
```

如果你是用 part2 或 part3 或 part4 挂载 U 盘的,也要卸载指定的设备文件，即

```
umount /dev/scsi/host0/bus0/target0/lun0/partx
```

注意：如果在 LINUX 下往 U 盘写文件后，在拔出 U 盘前一定要 umount 卸载或者在命令行下敲入以下命令确保 U 盘的缓冲被写回 U 盘物理存储区：

sync （回车）

如果不这样做，你写进 U 盘的内容并不一定会被真正写进 U 盘物理存储区

2.WINCE 部份

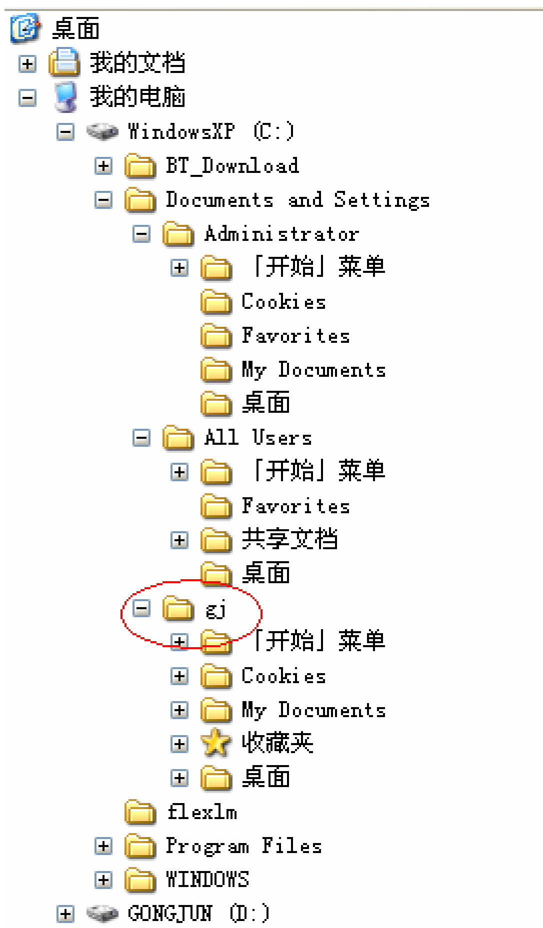
2.1 WINCE4.20 无法编译的解决方法

我们发现，部分用户在编译我们提供的 BSP 时出现编译出错的现象，无法编译。这种情况往往出现在正确配置 BSP，并开始编译后的几秒钟内。

出错信息在 Build.con 文件中可以找到，里面有类似“require 39 bytes memory”这样的错误提示信息。

经研究发现，出现这样的错误是因为 WINCE4.2 在编译的时候，需要在用户的系统文件夹中存取数据，如果用户的系统文件夹是中文名字，那么将会出现存取失败，导致编译错误。

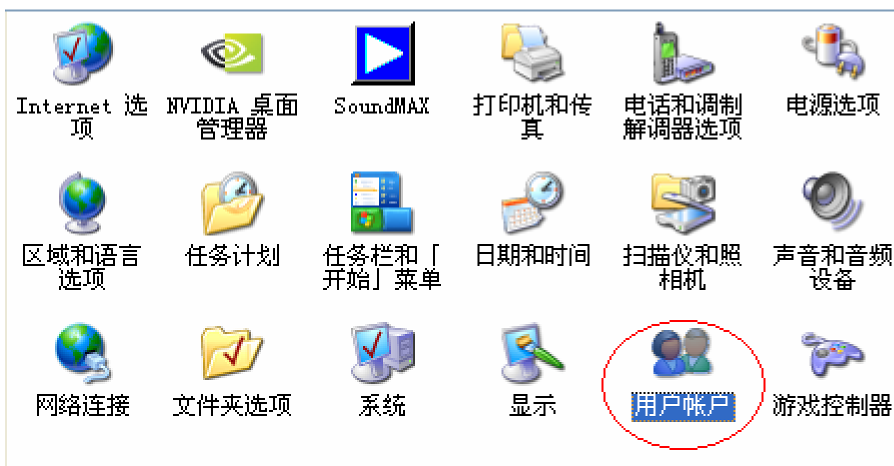
如果您的系统是 WindowsXP，而您的帐户是中文的，也就是下图中这个画红色圆圈的这个文件名字是中文的话，那么编译将会出错。



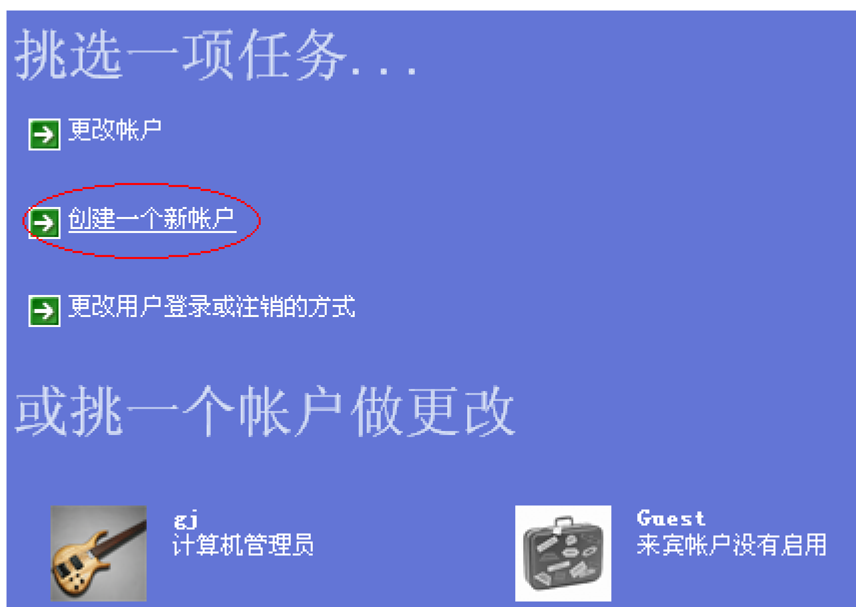
解决方案如下（以 WindowsXP 系统为例）：

首先要新建一个英文名字的管理员帐户（注意，仅只是将您的帐户名字更改为英文名字是没用的）；

在控制面板里面双击用户帐户：



单击“创建一个新帐户”新建一个管理员帐户：



给新帐户取一个名字（一定要用英文的名字!）：

为新帐户起名

为新帐户键入一个名称(T):

FS

这个名称会出现在 欢迎屏幕和「开始」菜单。

下一步(N) > 取消

选择帐户为“计算机管理员”:

挑选一个帐户类型

计算机管理员(A) 受限(L)

使用计算机管理员帐户，您可以:

- 创建、更改和删除帐户
- 进行系统范围的更改
- 安装程序并访问所有文件

< 上一步(B) 创建帐户(C) 取消

点击“创建帐户”就可以完成新建帐户;

然后再注销当前帐户，以刚才新建的帐户 ST 登陆:



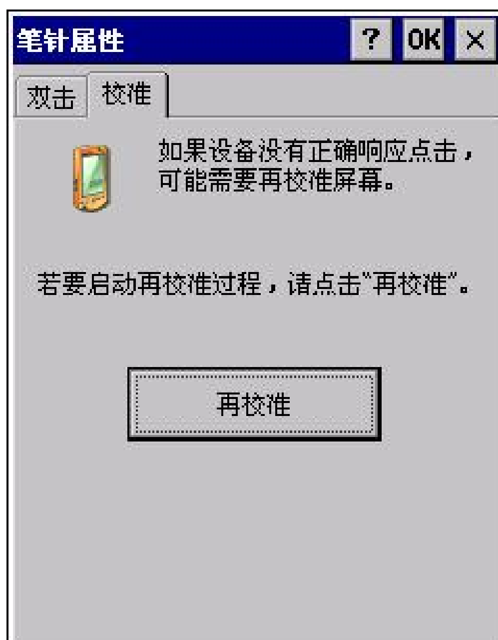
用新帐户登陆之后，删掉原来编译出错的 Project，按照说明书的步聚，先替换我们的 BSP，然后重新一步步建立新的 project，编译将不会出现错误了。

2.2 WINCE 下触摸屏的校验以及校验数据的保存

相同型号的 LCD 触摸屏，它们的校验数值也不是完全一样的，所以，每当用户拿到一个 LCD 触摸屏时，需要重新

使用触摸屏校验程序校验这个触摸屏，并把得到的校验数据永久保存到系统里面，这样每次系统重启时，不需要再次校验，就可以使用触摸笔很准确的点击目标位置。

在 WINCE 操作系统下，触摸屏的校验过程是这样的：用鼠标点击 WINCE 桌面图标【我的电脑】，然后在出现的窗口中点击图标【控制面板】，然后点击【笔针】，在弹出的窗口中，选择【校准】标签页，点击“再校准”按钮，如下图



在出现的校验画面中，我们可以看到中心位置有一个“＋”号，我们需要用触摸笔点击这个加号的中心位置，记住点击时，需要把触摸笔贴在这个中心位置 2，3 秒时间，再提起来，否则，如果轻触的时间太快，触摸程序可能捕捉不到用户的动作。当中心位置校验成功后，加号会移到左上角，用户需要跟着加号的移动，点击加号的中心位置，

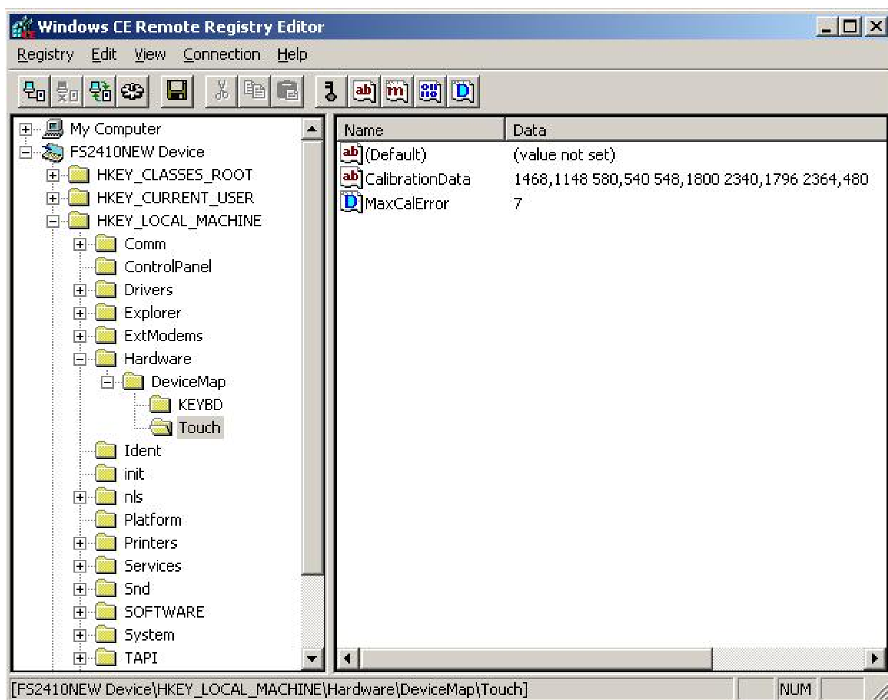
以最终完成整个校验过程。完成校验后，点击键盘回车键或鼠标左键确认校验结果。

当校验结果被确认后，校验数据就会被保存到 WINCE 的注册表当中，但是这并不是永久保存，因为数据并未被烧写到 FLASH 中，解决的办法是，将我们获取的校验数据加载到 PB 的平台的注册表中，然后重新编译平台，

生成包含触摸屏校验注册信息的 NK.nb0 映像。具体做法如下

通过 Microsoft ActiveSync 工具连接目标硬件平台（具体过程参见附录一的 6.1），然后在 PB 的 tools 菜单中点击“Remote Registry Editor”以运行远程注册表编辑工具查看 WINCE 的注册表内容，在注册表编辑器中打开

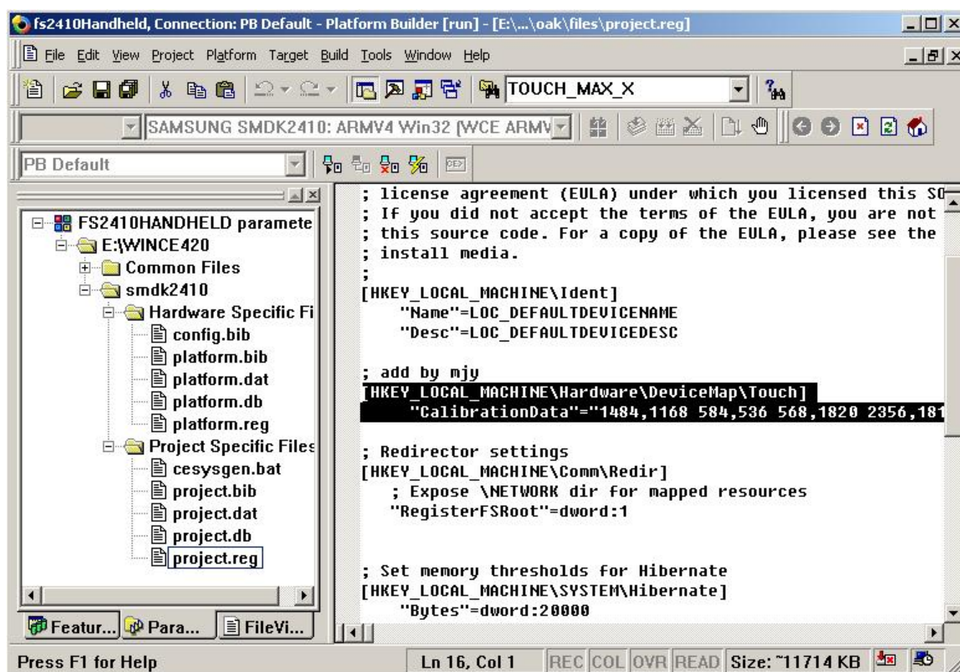
[HKEY_LOCAL_MACHINE\Hardware\DeviceMap\Touch]键，就可以看到触摸屏的校验数值，如下图



上图中的“CalibrationData”键值就是触摸屏的校验值，我们将这个值复制一下，然后打开 PB 的平台的 project.reg
文件，加入如下的触摸屏注册信息，其中"CalibrationData"=之后的值就是上面我们复制的触摸屏校验值。

```
[HKEY_LOCAL_MACHINE\Hardware\DeviceMap\Touch]
    "CalibrationData"="1484,1168 584,536 568,1820 2356,1812 2356,488"
```

添加后的结果如下图黑色区域所示：



添加完毕后，保存， 重新编译平台，生成新的映像文件，运行映像，就可以看到触摸屏校验信息已经在注册表里了。